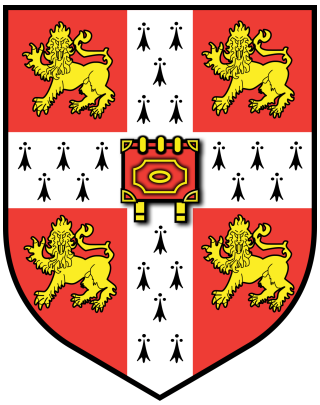
Deep Learning for Koopman Operator Optimal Control



Tom Xiaoding Lu
Pembroke College

May 17, 2020
Word Count: 7607 Figures Count: 75

I hereby declare that, except where specifically indicated, the work submitted herein is my own original work.

Signed: _____ Date: _____

Abstract

Tom Xiaoding Lu[1][2] Pembroke College

Nonlinear dynamics are ubiquitous in complex systems. Their applications range from robotics [1] to computational neuroscience [2]. They remain both theoretically and computationally challenging as there exists no generalised mathematical framework for their analysis.

In this work, we[3] explore the Koopman framework for globally linearising nonlinear dynamics. Under this framework, the nonlinear observable states are lifted into a higher dimensional, linear regime. The challenge is to identify the set of functions that facilitate the coordinate transformation to this lifted linear space. We tackle this problem using deep learning [3], where nonlinear dynamics are learnt in a model-free manner, i.e. the underlying dynamics are uncovered using data, rather than the nonlinear state-space equations.

Our main contributions include: an implementation of the Linearly Recurrent Encoder Network (*LREN*) that is faster than the existing implementation, and is significantly faster than the state-of-the-art deep learning based approach [4], requiring around 20 minutes to trained on a CPU, c.f. an hour on a GPU for the pendulum example [5]. We also propose a novel architecture termed Deep Encoder with Initial State Parameterisation (*DENIS*). By utilizing an auxiliary network parameterized by just the initial state of the system, *DENIS* is able to significantly outperform *LREN* in terms of long time-horizon prediction performance, and is on par with the state-of-the-art method by Lusche et al. [4]. Unlike their network, optimal control can be easily implemented through *DENIS*. By deriving the energy-budget control performance evaluation method, we demonstrate that *DENIS* also outperforms *LREN* in terms of control performance; it is also on par with and sometimes better than the iterative linear quadratic regulator (iLQR), which requires access to the state-space equations. Extensive experiments are done on *DENIS* in order

[1]Supervised by Dr Guillaume Hennequin

[2]Complete code and environment can be found at https://github.com/xl402/Deep-Koopman

[3]Refers to the joint work between me and Dr Hennequin

to validate its performance. We also describe another novel architecture termed Double Encoder for Input Non-affine systems (*DEINA*). We show the potential ability for *DEINA* to outperform existing Koopman frameworks for controlling input non-affine systems. We attribute this to using an auxiliary network to nonlinearly transform the inputs, thereby lifting the strong linear constraints imposed by the traditional Koopman approximation approach. Whenever possible, we show that our networks are able to discover the analytical Koopman eigenfunctions for certain dynamical systems; we visualise these eigenfunctions in section IV-A. Koopman model predictive control (KMPC) is implemented in order to verify that our models can also be successfully controlled under this popular approach.

The report structure is as follows: in section I, we introduce the mathematical definition of the Koopman framework, along with reviewing methods to approximate the Koopman eigenfunctions. Section II and III present the overall project set-up (training and analysis pipeline) and the dynamical systems considered in this work. Section IV details the *DENIS* architecture, and its performance comparison against *LREN* and iLQR. Ablation studies are also performed on the key aspects of *DENIS*. In section V, we present *DEINA* and show that it is able to discover analytically correct linearization of a nonlinear, input non-affine system. Finally, section VI concludes with a discussion of the limitations of the *DENIS* and *DEINA* architectures.

Contents

Nomenclature

Δt	Reciprocal of sampling frequency
u	System inputs
x	Observable states
y	Measurements on the state/ evaluations of the Koopman eigenfunctions
$\mathcal{A}$	Koopman operator
$\mathcal{L}$	Loss function/ components of the loss function
A	Koopman matrix (finite dimensional $\mathcal{A}$) and/or system matrix
B	Control matrix, mapping system inputs to changes in the observables
C_u	Input energy budget
F	Discrete dynamic map
f	Continuous dynamic map
$g()$	Coordinate transformation/ Koopman eigenfunctions
K	Feedback gain matrix, determining next time-step's inputs
k	Discrete time index
Q	State cost matrix
R	Controller input cost matrix
t	Continuous time index
DEINA	Double encoder for input non-affine systems
DENIS	Deep encoder with initial state parameterisation
LREN	Linearly recurrent encoder network
DL	Deep learning
DMD	Dynamic mode decomposition
iLQR	Iterative linear quadratic regulator
KMPC	Koopman Model predictive control
LQR	Linear quadratic regulator
MAD	Median of absolute deviation
MPC	Model predictive control
MSE	Mean squared error

I. Introduction

Nonlinear dynamics are ubiquitous in complex systems. Their applications are commonly found in a variety of disciplines such as robotics [1], systems biology [6], industrial plant control [7], computational neuroscience [2] and financial modelling [8]. They remain both theoretically and computationally challenging, as there exists no generalised mathematical framework for their analysis [9]. The problem becomes even more complicated when the system dynamics are unknown, and the intrinsic states cannot be observed.

The Koopman [10] framework is becoming increasingly popular for obtaining linear representations of nonlinear systems from data, thereby enabling comprehensive techniques for linear system analysis and control. Much of the recent research directly seeks eigenfunctions of the Koopman operator [4], as they provide coordinates which globally linearise the nonlinear dynamics. This process is inherently data-driven and benefits from the power of deep learning (DL) [3]. Many works also address the subject of optimal control under the Koopman framework [11]–[13], although very few are based on DL approaches.

This section begins by providing an overview of the Koopman mathematical framework, we then review and categorise methods for Koopman eigenfunction discovery found in literature. Finally, the shortcomings of the existing methods regarding both system dynamics reconstruction and optimal control are discussed.

A. Koopman Operator Overview

We may consider a dynamical system, in an abstract sense, to be composed by two things: a set of *states* through which we can describe the evolution of the system, and a set of *rules* for that evolution. A prominent mathematical interpretation of which considers a set of coupled differential equations that is represented as:

$$\dot{\mathbf{x}} = f(\mathbf{x}) \tag{I.1}$$

where $\mathbf{x}$ belongs to the state space $S \in \mathbb{R}^n$ and $f : S \to S$ is a vector field on that state space. We may also consider dynamical systems given by the discrete-time map:

$$\mathbf{x}_{k+1} = F(\mathbf{x}_k) \tag{I.2}$$

where $\mathbf{x}$ belongs to the state space $S \in \mathbb{R}^n$ with $F : S \to S$ being the dynamic map, and $k \in \mathbb{Z}$ denotes the discrete time-index. In both continuous and discrete-time cases, the

dynamic maps are generally nonlinear and are often unknown.

The traditional approach for studying unknown systems begins by constructing a model in the form of *I.1* or *I.2*. These approximated systems are either solved analytically or numerically in order to analyse the dynamics, which often attributes to finding the attractors and invariant sets, etc. The issue of the traditional approach is that, real-world systems are high-dimensional, hence simulating the evolution of trajectories for these systems are prohibitively expensive [14]. As a result, the field of dynamical analysis has started shifting from *model-driven* to a more *data-driven* approach. This shift is further progressed by the vast amount of data that is readily available.

In 1931, B.O. Koopman described dynamical systems in terms of the evolution of functions in the Hilbert space $g(\mathbf{x})$ [10], with $g : S \to \mathbb{C}$, which are complex-valued measurements of the states rather than the states themselves. Consider the discrete-time dynamical system in equation *I.2*, the Koopman operator $\mathcal{A}$ advances these measurements one step forward in time:

$$\mathcal{A}g\left(\mathbf{x}_k\right) = g\left(\mathbf{x}_{k+1}\right) \tag{I.3}$$

as $g\left(\mathbf{x}_{k+1}\right) = g\left(F\left(\mathbf{x}_k\right)\right)$. Equation *I.3* can be re-written as:

$$\mathcal{A}g = g \circ F \tag{I.4}$$

where $\circ$ denotes the function composition operator, therefore $\mathcal{A}$ is a linear operator, with which the analysis of nonlinear dynamics can be lifted into a linear regime. This generic Koopman framework is shown in figure 1(a).

In order to find a finite-dimensional representation of $\mathcal{A}$, one may seek the Koopman eigenfunctions $\{\phi_1, \phi_2, \ldots\}$ and eigenvalues $\{\lambda_1, \lambda_2, \ldots\}$ which span the Koopman invariant subspace $\mathcal{U}$. All measurements coming from $\mathcal{U}$ remain within the subspace under the Koopman operator such that:

$$g(\mathbf{x}) = \sum_{k=0}^{\infty} \alpha_k \phi_k(x) \tag{I.5}$$

$$\mathcal{A}g(\mathbf{x}) = \sum_{k=0}^{\infty} \beta_k \phi_k(x) \tag{I.6}$$

where $\beta_k = \alpha_k \lambda_k$. When a finite number m of such eigenfunctions is sought, the Koopman operator becomes a Koopman matrix $A \in \mathbb{C}^{m \times m}$. The key in applying Koopman analysis

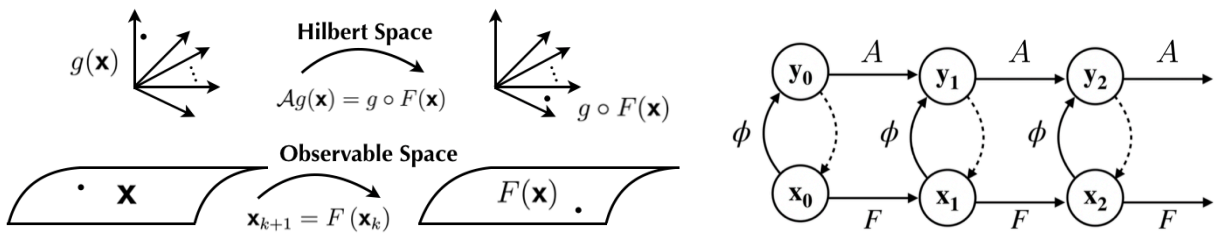


Fig. I.1: Pictorial representation of the Koopman framework. Left: Koopman viewpoint lifts the dynamics from the observable space to the Hilbert space, where dynamics are linear but infinite. Right: the representation becomes finite when a finite number of Koopman eigenfunctions ϕ are sought, the Koopman operator $\mathcal{A}$ becomes a Koopman matrix A. Note g denotes all possible measurements on $\mathbf{x}$, whereas $\mathbf{y}$ denotes the set of eigenfunctions that can be used to span the Koopman space.

is to directly obtain a finite number of exact or approximate Koopman eigenfunctions, i.e. $\mathbf{y} = \phi(\mathbf{x})$, $\phi : \mathbb{R}^n \to \mathbb{C}^m$ and $\mathbf{y}_{k+1} = A\mathbf{y}_k$. This concept is illustrated in figure 1(b). If ϕ are invertable, then $\mathbf{x}_{k+1}$ can be reconstructed from $\mathbf{y}_{k+1}$.

B. Koopman Operator Optimal Control Overview

There are several ways of generalising the Koopman operator to control systems [15]–[17]. We begin by considering a discrete-time nonlinear controlled dynamical system:

$$\mathbf{x}_{k+1} = F(\mathbf{x}_k, \mathbf{u}_k) \tag{I.7}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state of the system, $u \in \mathcal{U} \subset \mathbb{R}^m$. Korda and Mezic [15] have shown that a set of linear predictors $\mathbf{y} = \phi(\mathbf{x}), \phi : \mathbb{R}^n \to \mathbb{R}^l$ can be constructed such that:

$$\mathbf{y}_{k+1} = A\mathbf{y}_k + B\mathbf{u}_k \tag{I.8}$$

$$\hat{\mathbf{x}}_{k+1} = C\mathbf{y}_{k+1} \tag{I.9}$$

equations *I.8* and *I.9* indicate an optimal controller may be designed for $\mathbf{y}$, which is equivalent to achieving *Koopman operator optimal control* (KOOC) in $\mathbf{x}$ in the limit that $\hat{\mathbf{x}}_{k+1} = \mathbf{x}_k$.

This problem setup corresponds to a Koopman operator $\mathcal{A} = [A, B]$ operating on the extended state-space $\mathcal{X} = [\mathbf{x}, \mathbf{u}]^T$ such that:

$$\mathcal{A}\phi\left(\mathcal{X}_k\right) = \phi\left(\mathcal{F}\left(\mathcal{X}_k\right)\right) \tag{I.10}$$

where $\mathcal{F}\left(\mathcal{X}_k\right) = \left[F\left(\mathbf{x}_k, \mathbf{u}_k\right), \mathbf{u}_{k+1}\right]^T$, equation *I.8* is achieved by constraining ϕ such that:

$$\phi(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \psi(\mathbf{x}) \\ \mathbf{u} \end{bmatrix} \tag{I.11}$$

In section V, we will see that the strong linear constraint on $\mathbf{u}$ imposed by equation *I.11* maybe lifted to achieve better predictive performance.

Equation *I.7* warrants two families of non-linear systems: *control affine* and *control non-affine*. In this work, we consider these two cases separately. The motivation behind this is that, for control affine systems:

$$\mathbf{x}_{k+1} = F(\mathbf{x}_k) + \tilde{B}\mathbf{u}_k \tag{I.12}$$

we may first discover the Koopman eigenfunctions $\mathbf{y}$ for the autonomous dynamics (i.e. $\mathbf{u}_k = 0, \ \ \forall k \in \mathbb{Z}$). We can then fix $\mathbf{y}$ and discover $\tilde{B}$. This subtlety gives rise to one of our main contributions *DENIS*, which will be described in depth in section IV-B.

C. Existing Methods for Koopman Eigenfunction Discovery

Recent advances in computational methods enabled the data-driven discovery of Koopman eigenfunctions [4]. There are two classes of methods: the *dynamic mode decomposition* (DMD) [18] class and the *Deep Learning* (DL) class. We will later see how the DL class closely relates to the DMD class. This section aims to provide a high-level overview on prominent methods within these two classes, while also empirically categorize and rank them in terms of their predictive and control performance.

Dynamic mode decomposition (DMD) is rooted in the analysis of fluid models, where snapshots of state measurements are assembled into a matrix:

$$\mathbf{X}_i = \begin{pmatrix} | & | & & | \\ g\left(\mathbf{x}_i\right) & g\left(\mathbf{x}_{i+1}\right) & \cdots & g\left(\mathbf{x}_{M+i}\right) \\ | & | & & | \end{pmatrix} \tag{I.13}$$

The algorithm estimates a linear relationship between two data matricies:

$$\tilde{A} = \mathbf{X}_0^{\dagger}\mathbf{X}_1 \tag{I.14}$$

where $\tilde{A} \in \mathbb{R}^{n \times n}$ is the propagator matrix, and $\dagger$ denotes the pseudo-inverse. When the measurement function $g(\mathbf{x})$ is the identity mapping, i.e. $g(\mathbf{x}) = \mathbf{x}$, for linear systems, the eigenvalues of $\tilde{A}$ and the Koopman operator are identical. For nonlinear systems, one can work with an augmented state consisting of possibly nonlinear *dictionary* functions g of the state, this procedure is termed *extended dynamic mode decomposition* (eDMD) [19]. We can perform matrix diagonalization:

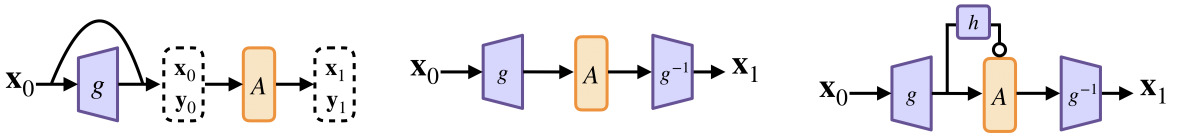
$$\tilde{A} = P^{-1} A P \tag{I.15}$$

where A forms an approximation of the Koopman operator, and $P^{-1}g$ are the corresponding approximated Koopman eigenfunctions. As we are given the propagator matrix $\tilde{A}$, we can always construct the Koopman operator A and its eigenfunctions, with a slight abuse of notation. In the remaining work, we do not explicitly distinguish between $\tilde{A}$ and A, i.e. the propagator will be viewed as a Koopman operator. The performance of eDMD depends strongly on the choice of the *dictionary* functions g, which usually requires prior knowledge of the system. Compact dictionaries with high capacities can be achieved by formulating eDMD as a kernel method, which is termed kDMD [20]. However, the choice of kernels is empirical, and the algorithm may overfit the data.

The approaches above may be further improved by adding in *delayed coordinates*, the motivation is that, A should not only be fitted to predict one time-step transition of the measurements (equation *I.15*), but also multiple or preferably infinite time-steps. We therefore create the following time-delayed snapshot matrix:

$$H = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_p \\ \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_{p+1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_q & \mathbf{x}_{q+1} & \cdots & \mathbf{x}_{q+p-1} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 & A\mathbf{x}_1 & \cdots & A^{p-1}\mathbf{x}_1 \\ A\mathbf{x}_1 & A^2\mathbf{x}_1 & \cdots & A^p\mathbf{x}_1 \\ \vdots & \vdots & \ddots & \vdots \\ A^{q-1}\mathbf{x}_1 & A^1\mathbf{x}_1 & \cdots & A^{q+p}\mathbf{x}_1 \end{bmatrix} \tag{I.16}$$

where $q, p \in \mathbb{Z}$. Through singular value decomposition of H we may discover the Koopman operator A and its eigenfunctions, this method is termed the *time-delayed DMD* (tDMD) [21]. Because of the large size of this system of equations, iterative methods [22] together with sparse regularized approaches [23] have been popular in obtaining approximated solutions. Deep learning based dictionary discovery requires no prior knowledge of the system, and dictionary sparsity is easily implemented through regularization. Furthermore, by stepping



(a) Linearly Recurrent Encoder *LREN* (b) Encoder Decoder (c) Parameterized Encoder Decoder

Fig. I.2: Existing network architectures found in literature, all networks are designed to only reconstruct autonomous trajectories

several time-steps into the model architecture, we can mimic the *delayed coordinates* formulation introduced by *Hankel* matrices efficiently (further discussed in section IV-A).
There are three types of neural network architectures that are found across literature for discovering the Koopman eigenfunctions (shown in figure I.2). We found that all architectures are used only for learning autonomous trajectories, hence by construction, cannot handle control-non-affine systems. This motivates our work on *DEINA* described in section V. The key discrepancy between these architectures is whether $\mathbf{x}$ can be linearly reconstructed from the Koopman eigenfunctions $\mathbf{y}$, specifically whether equation *I.9* is valid. While this point may be subtle, it determines whether KOOC can be trivially achieved through methods such as Linear Quadratic Regulator [24], or requires more sophisticated methods such as Koopman Model Predictive Control [25]. Concretely, we note that if $\mathbf{x}_k = C\mathbf{y}_k$, for the Koopman framework described by equations *I.8* and *I.9*, representing costs associated with our observable states $\mathbf{x}$ and inputs $\mathbf{u}$ as Q and R respectively, KOOC is achieved by solving a discrete-time algebraic Riccati equation (DARE) with an augmented cost $\tilde{Q} = C^T Q C$.
Because it is desirable for the observables $\mathbf{x}$ to be within the span formed by $\mathbf{y}$, a type of networks termed *Linearly Recurrent Encoder Network* (LREN) [5] shown in figure 2(a), adds skip-connections so that $\mathbf{y} = [\mathbf{x} \quad g(\mathbf{x})]^T$. This ensures that observables $\mathbf{x}$ are within the measurement functions and are advanced linearly by A. The downside of this approach is that the representation power is limited by the fact that $\mathbf{x}$ is linearly coupled with the measurements $g(\mathbf{x})$, hence a high latent dimension (number of measurements) may be required to gain good model performance.

Other architectures relax the constraint by allowing $\mathbf{x}$ to be non-linearly reconstructed from the measurements $g(\mathbf{x})$, [26] has shown the proposed architecture in 2(b) achieves

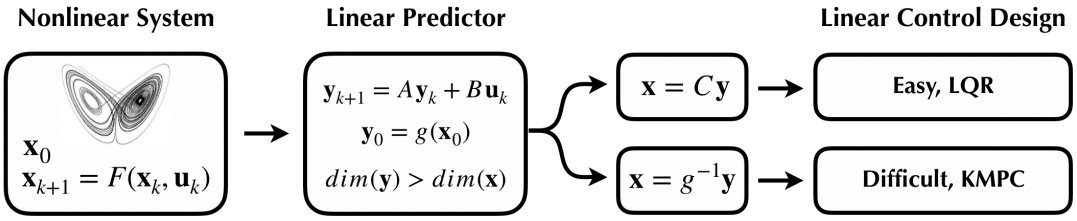


Fig. I.3: Overall existing workflows for controlling nonlinear systems under Koopman framework. Left to right: start with a nonlinear system, construct models to find linear representation of the system in a lifted dimension; if **x** is in the span of **y** then KOOC can be easily achieved with LQR, otherwise more sophisticated methods are required

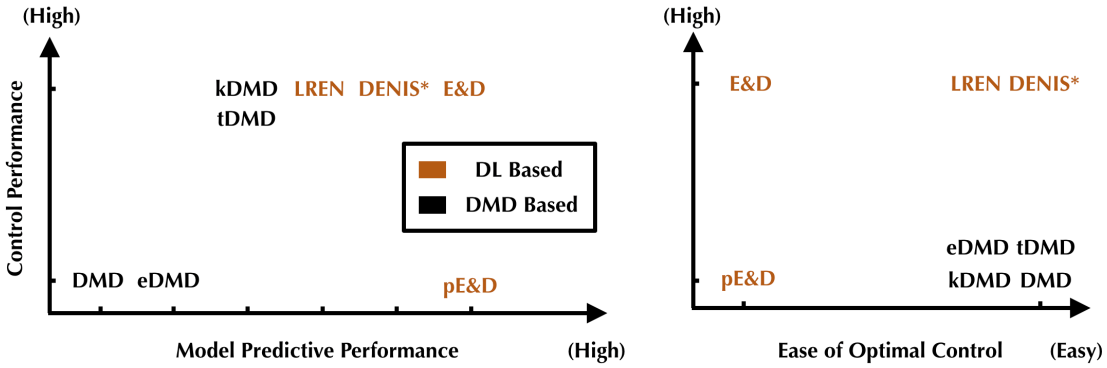


Fig. I.4: Empirical comparison between different methods for learning Koopman eigenfunctions found in literature. Left: control performance (measured by how extensive the model is tested on closed-loop systems) against model predictive performance (measured by how well model is able to reconstruct the dynamics). Right: control performance against ease of optimal control (split between 'difficult' and 'easy', where 'difficult' requires more sophisticated control methods such as Koopman Model Predictive Control)

good performance while having a compact latent space. Lusch et al. [4] further improves the representational capacity of the network by using an auxiliary network to parameterize the Koopman operator A (shown in figure 2(c)). Thereby allowing the network to learn the exact representation of certain dynamics which exhibit a *continuous Koopman Spectrum*, i.e. state dependent Koopman eigenvalues. The issue with these two approaches is that KOOC becomes more complicated, and one can argue that the learnt system does not evolve linearly under controller inputs. This may be addressed by using Koopman Model Predictive Control, which solves a convex optimization problem within each time-step to

achieve control, which is less elegant and in many cases, sub-optimal. For architecture 2(c), optimal control becomes even more obscure due to the state dependency of A.
It is therefore interesting to observe the trade off between *model predictive* performance and *optimal controllability* performance for deep learning based approaches. One naturally expects the closed-loop performance to increase as the reconstruction error of the modeled system trajectories decrease, however, as models become more contrived, methods to achieve optimal control become more obscure and may in-turn hinder control performance. We also note that for all DMD based approaches, KOOC is readily implemented as one can easily include $\mathbf{x}$ in the measurements. Figure I.4 empirically summarizes the existing methods found in literature. For motivation, we also include our contribution *DENIS* for comparison, where we show in sections IV-D and IV-E that it achieves both good control and predictive performance, while KOOC can be easily implemented with LQR.

II. Project Setup

Given the goal of this work is to achieve *principled* methods for predicting non-linear dynamics, model validation and ablation studies are essential for verifying our assumptions made on certain architectures. As mentioned in section I-C, there are two aspects of model validation/comparison: *model predictive performance*, commonly measured by the mean squared error on the predicted trajectories, and *optimal control performance*, which is more subtle to measure (discussed more in section IV-E). Because of this, our work requires a more contrived validation pipeline than other works in the field of deep learning. Furthermore, because we are taking the deep learning approach, even the simplest architectural changes require time to compute results, it is therefore essential to formulate a research framework which ensures:

- Rapid model prototyping and local training
- Parallel training for multiple experiments on the cloud
- Shared framework for model comparison

This section describes the high-level framework for program setup and tools used in developing this project.

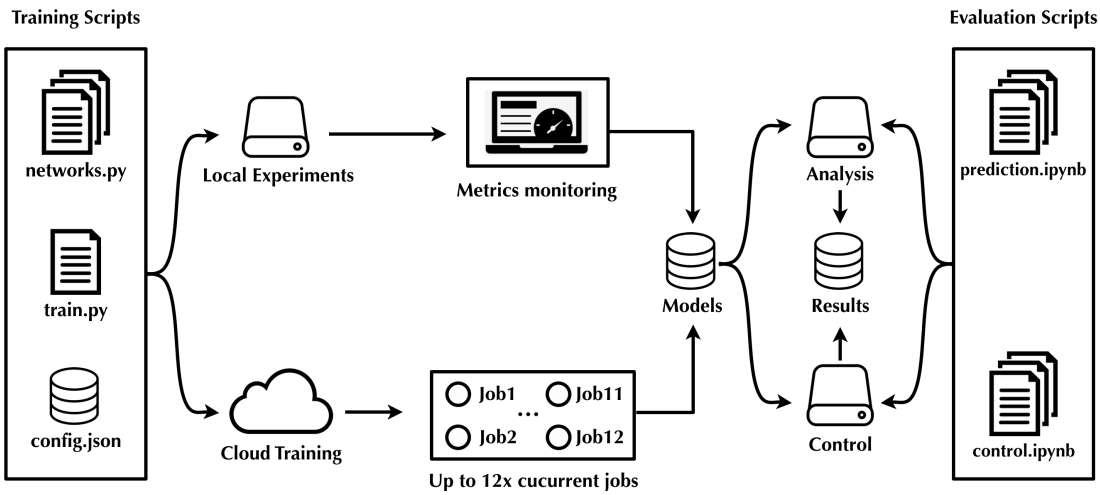


Fig. II.1: Project workflow overview

A. *Project Tools and Packages*

Unless explicitly mentioned, we re-implement all existing model architectures and control methods found in literature from scratch. This ensures existing architectures are constructed so that their outputs fall under our evaluation methods, they can then be fairly compared against our proposed architectures.

The majority of the project was implemented in *Python*, except for a few cases where *MATLAB*'s control toolbox was utilised. Models were initially written in *TensorFlow* [27], however, due to *FloydHub*'s initial incompatibility with *TensorFlow 2.0*, all programs were then translated into *PyTorch* [28]. We also utilise popular packages such as *SciPy* [29] and *Harold* [30] for model evaluation. Koopman Model Predictive Control (KMPC) was coded from scratch, with the *CVXPY* [31], [32] toolbox utilised for solving the associated convex optimization problem.

B. *Project Workflow*

The project makes a clear distinction between training and evaluation scripts. Training scripts are written with a minimum number of dependencies, only low-level *Numpy* and *PyTorch* are used. This ensures our networks and training scripts can be easily run by the public. The training script is fully configured by a *JSON* file, i.e. *all* networks and their configurations discussed in this project may be trained with only one training script. In order to achieve this, we aimed to write generic network blocks, together with generic loss

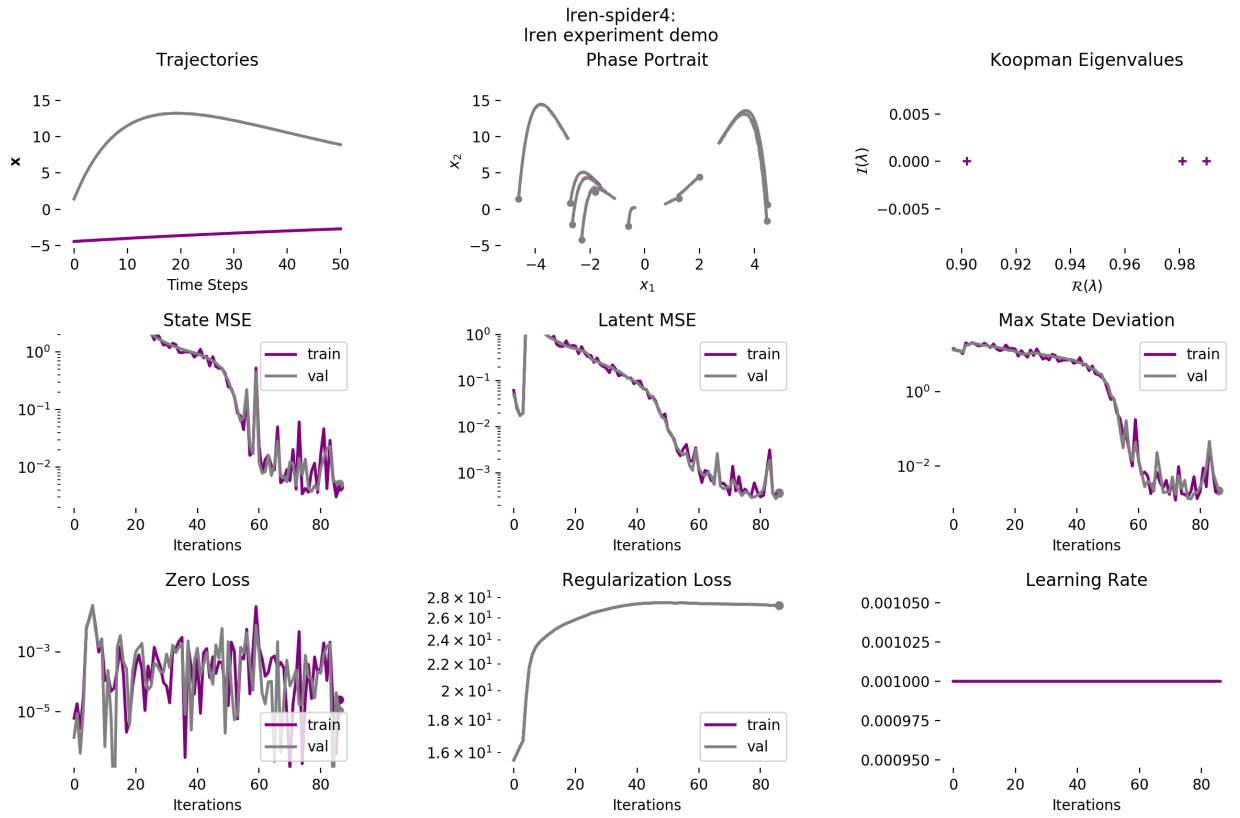


Fig. II.2: Graphical interface during local training, metrics tracked include the five components of the loss function and learning rate

functions, that can be configured according to the *JSON* file.

Training can be conducted locally or on the cloud. Local training is commonly used for quick model debugging when developing new architectures. A custom graphical interface is shown in figure II.2, where all key metrics such as components of the loss function and predicted trajectories may be visualised as live updates. Once the model is verified to be correct, both the model and the training script are submitted to *Floydhub*, where up to 12 experiments may be conducted simultaneously. This greatly reduces the wait time between changing model architecture and obtaining results corresponding to those changes.

Finally, those locally or cloud-trained models are fed into local evaluation scripts. These scripts typically use more niche packages such as *CVXPY* and *Harold*, since our model validation is specific to this project. The outputs of figures/simulation videos are then saved locally, completing a cycle of model development.

III. Nonlinear Systems

In order to be able to systematically compare our network against those within literature, we utilise the 3 examples presented by Lusch et al. [4]. We found that there is a lack of systematic comparison for deep-learning (DL) based Koopman frameworks; this is true for dynamics prediction performance comparison, and also for optimal control performance analysis. In fact, we have found no paper comparing the control performance for different DL Koopman architectures. We therefore introduce another dynamical system - the *Duffing Oscillator*, a more challenging system to be controlled. This section describes the detailed dynamics of all four dynamical systems considered.

A. Discrete Koopman Spectrum Example

The first toy-example we consider has the following continuous dynamics:

$$\begin{aligned} \dot{x}_1 &= \mu x_1 \\ \dot{x}_2 &= \lambda \left(x_2 - x_1^2 \right) \end{aligned} \tag{III.1}$$

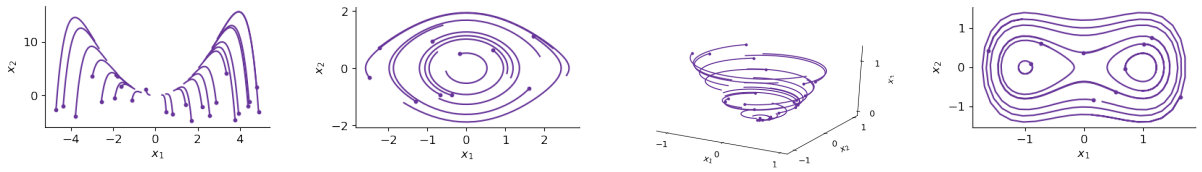
We term this dynamical system "spider", with the phase-portrait shown in figure 1(a). This system has been studied extensively in literature [21], [24], [33], and is normally used as a toy example for teaching the Koopman framework. It has analytical Koopman eigenfunctions. Consider a coordinate transformation where $\mathbf{y} = [x_1 \; x_2 \; x_1^2]^T$. Then by applying chain rule:

$$\dot{\mathbf{y}} = \dot{\mathbf{x}} \frac{d\mathbf{y}}{d\mathbf{x}} = \begin{bmatrix} \mu & 0 & 0 \\ 0 & \lambda & -\lambda \\ 0 & 0 & 2\mu \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = A\mathbf{y} \tag{III.2}$$

We see that under this simple coordinate transformation, our initially non-linear system shown in equation III.1 has become linear. The eigenvalues and their corresponding eigenvectors associated with matrix A are μ, 2μ, λ and $[1 \; 0 \; 0]^T$, $[0 \; 0 \; 1]^T$, $[0 \; 1 \; b]^T$, with $b = \dfrac{\lambda}{\lambda - 2\mu}$. Therefore the associated Koopman eigenfunctions are obtained by dotting our coordinate transformation with these eigenvectors:

$$\phi_1 = x_1, \quad \text{and} \quad \phi_2 = x_2 - b x_1^2 \quad \text{with} \tag{III.3}$$

Note x_2 is not present as we can non-linearly reconstruct our observables using ϕ_1 and ϕ_2, hence it is not necessary for it to be included in the Koopman invariant subspace.
We select $\mu = -0.05$ and $\lambda = -1$ for this dynamic system, which is identical to the values used by Lusch et al.



(a) Spider (b) Pendulum (c) 3D Fluid flow (d) Duffing oscillator

Fig. III.1: Phase portrait for four different systems. Solid dots indicate initial conditions, trajectories randomly sampled from the validation data

B. Continuous Koopman Spectrum Examples

The major contribution brought forward by Lusch et al. is the fact that their model (basic architecture shown in figure 2(c)) is able to discover systems exhibiting *continuous Koopman Spectrum*. This means the Koopman eigenvalues are state-dependent and vary continuously. One such system is the non-linear pendulum described by:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\sin(x_1) \end{aligned} \tag{III.4}$$

with sampled trajectories shown in figure 1(b). In this system, the eigenvalue spectrum changes as the energy of the pendulum is increased, posing a significant challenge for classical Koopman embedding techniques. In the example, we consider the frictionless pendulum, so that the system is conservative and trajectories evolve on Hamiltonian energy level sets. One intuitive Koopman eigenfunction is the Hamiltonian energy of the system:

$$\frac{1}{2}x_2^2 - \cos(x_1) \tag{III.5}$$

with a unit corresponding eigenvalue, as it is preserved throughout the dynamics.
One other system under consideration is the mean-field model [34] for the fluid flow past a circular cylinder at Renolds number 100:

$$\begin{aligned} \dot{x}_1 &= \mu x_1 - \omega x_2 + A x_1 x_3 \\ \dot{x}_2 &= \omega x_1 + \mu x_2 + A x_2 x_3 \\ \dot{x}_3 &= -\lambda\left(x_3 - x_1^2 - x_2^2\right) \end{aligned} \tag{III.6}$$

where $\mu = 0.1, \omega = 1, A = -0.1,$ and $\lambda = 10$, shown in figure 1(c). This system is a challenging canonical system in fluid dynamics, and is also investigated by Lusche et al.

The third system with continuous Koopman spectrum is the *duffing oscillator* described by:

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= x_1 - x_1^3 \end{aligned} \tag{III.7}$$

which has been used to compare different Koopman model predictive control frameworks [12], [35]–[37], however, was not used by any deep learning based methods. This system is interesting as from figure 1(d), the uncontrolled system has two stable equilibria at $[-0.5\ 0]^T$ and $[0.5\ 0]^T$ as well as an unstable equilibrium at the origin.

We constrain the Hamiltonian energy of these systems to be unity during data generation. All dynamics are numerically integrated with *Scipy*'s *odeint* which uses a LSODA solver [38]. Sampling frequencies are shown in table I, systems are initialised with 20,000 random initial conditions; 70%-30% train validation split ratio was also utilised.

IV. *DENIS*: Deep Encoder Network with Initial State Parameterisation

In this section, we present our network *DENIS* (**D**eep **e**ncoder **n**etwork with **i**nitial **s**tate **p**arameterisation). In order to demonstrate the motivation behind our network, we reproduce results obtained by Lusch et al. [4] and also implement a network architecture termed *LREN* (**l**inearly **r**ecurrent **e**ncoder **n**etwork) similar to Otto et al. [5].

We compare our networks against the existing architectures by using systems described in section III. We are able to demonstrate that *DENIS* has the ability to discover analytical Koopman eigenfunctions for systems with discrete Koopman spectrum; it is also able to form better approximations for systems with continuous Koopman spectrum, and its prediction performance is comparable with the state-of-the-art framework [4]. We also show that under our framework, optimal control of non-linear dynamics is a trivial extension of implementing a Linear Quadratic Regulator, whereas frameworks that identify the exact Koopman eigenfunctions do not have obvious/analytically correct methods for optimal control. We also show that our implementation of *LREN* is significantly faster to train than the existing architectures.

A. Review of LREN: Linearly Recurrent Encoder Network

While the architecture behind the *Linearly recurrent encoder network* (*LREN*) is not new and was first proposed by Otto et al. [5]. There exists no comprehensive review for such system in terms of:

- Whether the model is able to correctly discover Koopman eigenfunctions for systems with analytical solutions to the Koopman framework
- effect of model hyper-parameters on its prediction performance
- its control performance across a variety of systems

In this work, we re-implement a version of *LREN* and task it to reconstruct all system dynamics described in section III, thereby providing a comprehensive study of such framework.

The principal behind the *LREN* architecture is that: in order to apply Koopman Operator Optimal Control (KOOC), we need to identify a Koopman Invariant Subspace, where the observable states are within this subspace. The simplest constraint we can exert on the solution space is to let $\mathbf{y} = [\mathbf{x}\ \mathbf{z}]$, $\mathbf{z} = g(\mathbf{x})$, where $\mathbf{y}$ are coordinates which can be linearly evolved with a matrix A. As mentioned in section I-C, it is incorrect to call A and $\mathbf{y}$ the Koopman operator and Koopman eigenfunctions respectively, since we need to ensure A is in diagonal form. As we can always do the diagonalisation post-hoc, we may refer A as the Koopman operator, even though this is strictly speaking incorrect. Otto et al. did not explicitly state the mathematical framework behind the *LREN* architecture, here we provide our view on how *LREN* may be framed as a non-convex optimisation problem: treating dynamical system's initial conditions $\mathbf{x}_0$ as a random variable, our goal is to minimize the following objective function:

$$\min_{A,\,g} = \mathbb{E}_{\sim \mathbf{x}_0}\left[\sum_{k=1}^{T} A^k g(\mathbf{x}_0) - g(\mathbf{x}_k)\right] \tag{IV.1}$$

$$\text{s.j.} \quad g(\mathbf{x}) = [\mathbf{x} \quad \mathbf{z}]^T, \quad \forall\ \mathbf{z} \in \mathbb{R}^M \tag{IV.2}$$

$$\mathbf{x}_k = \mathbf{x}_0 + \int_{t=0}^{k\Delta t} f(\mathbf{x}_0)dt \tag{IV.3}$$

where Δt denotes the sampling frequency of the continuous-time system, and f denotes the non-linear flow map of such system. From equation *IV.1*, we can see the global minimum is met when A advances $g(\mathbf{x})$ in a linear fashion, where $\mathbf{x}$ are evolved on the discrete-time flow map F where:

$$F(\mathbf{x}_k) = \mathbf{x}_0 + \int_{t=0}^{k\Delta t} f(\mathbf{x}_0)dt \tag{IV.4}$$

This highly non-linear optimisation problem may be solved using a neural network, $g(\mathbf{x})$ can be interpreted as an unconventional "encoder" which expands system's dimension.

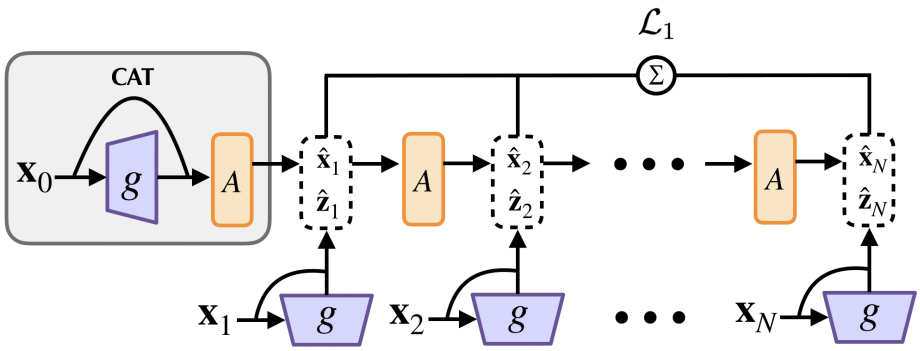


Fig. IV.1: Unrolled architecture of a Linear Recurrent Encoder cell, region with grey background indicates the basic unit of the architecture

A skip connection in the form of a concatenation can be used to enforce constraint *IV.2*. The multiple time-steps from $k = 1$ to T may be performed by simply linearly unrolling the network dynamics. Figure IV.1 demonstrates the equivalent neural network architecture, where the grey background region indicates the entire architecture of the network. Remaining figure merely demonstrates how the dynamics may be unrolled to obtain the objective function in equation *IV.1*. We would like to draw the link between the network architecture and method studied in the *Hankel alternative view of Koopman* (HAVOK) [21]. Which when compared against DMD, achieves higher performance as the Koopman operator is fitted on multiple time-step evolution. We will show in section IV-E that the ability to correctly predict over longer time horizon is essential to performing control on the system.

In order to complete network definition, we need to translate objective *IV.1* into a loss function. Inspired by [4], we consider five components within our loss function:

$$\mathcal{L} = \sum_{i=1}^{5} \alpha_i \mathcal{L}_i \tag{IV.5}$$

with α_i being hyperparameters controlling the relative importance across different components. Let b and k denote indices of positions within the mini-batch and time-step respectively. $\mathcal{L}_i$ are broken down into:

1) *Weighted-average state reconstruction loss*

$$\mathcal{L}_1 = \frac{1}{TB\sum_k c_k} \sum_{b=1}^{B} \sum_{k=1}^{T} c_k \left\| \hat{\mathbf{x}}_{k,b} - \mathbf{x}_{k,b} \right\|_2 \tag{IV.6}$$

with c_k being a positive, non-increasing sequence of weights, this can be set to all ones if all time-steps are deemed equally important. We found through training that this typically encourages smoother predictions. T and B correspond to maximum prediction time horizon and mini-batch size respectively.

2) *Weighted-average linear dynamics loss*

$$\mathcal{L}_2 = \frac{1}{TB\sum_k c_k} \sum_{b=1}^{B} \sum_{k=1}^{T} c_k \left\| \hat{\mathbf{y}}_{k,b} - \mathbf{y}_{k,b} \right\|_2 \tag{IV.7}$$

To discover Koopman eigenfunctions, we learn the linear dynamics A on the latent space. This ensures that the latent space gets linearly evolved. Just like equation *IV.6*, they can also be weighted according to their positions within the time index. The weights should be identical to those set in equation *IV.6*.

3) *Maximum reconstruction error loss*

$$\mathcal{L}_3 = \frac{1}{B} \sum_{b=1}^{B} \left\| \left\| \hat{\mathbf{x}}_{k,b} - \mathbf{x}_{k,b} \right\|_2 \right\|_\infty \tag{IV.8}$$

here we penalise the worst performing prediction within the mini-batch

4) *Zero loss*

$$\mathcal{L}_4 = \left\| g(0) \right\|_2 \tag{IV.9}$$

This loss function is added to ensure zero in the observable gets mapped to zero in the latent coordinates, we will show in section IV-F that this ensures control stability around the fixed-point.

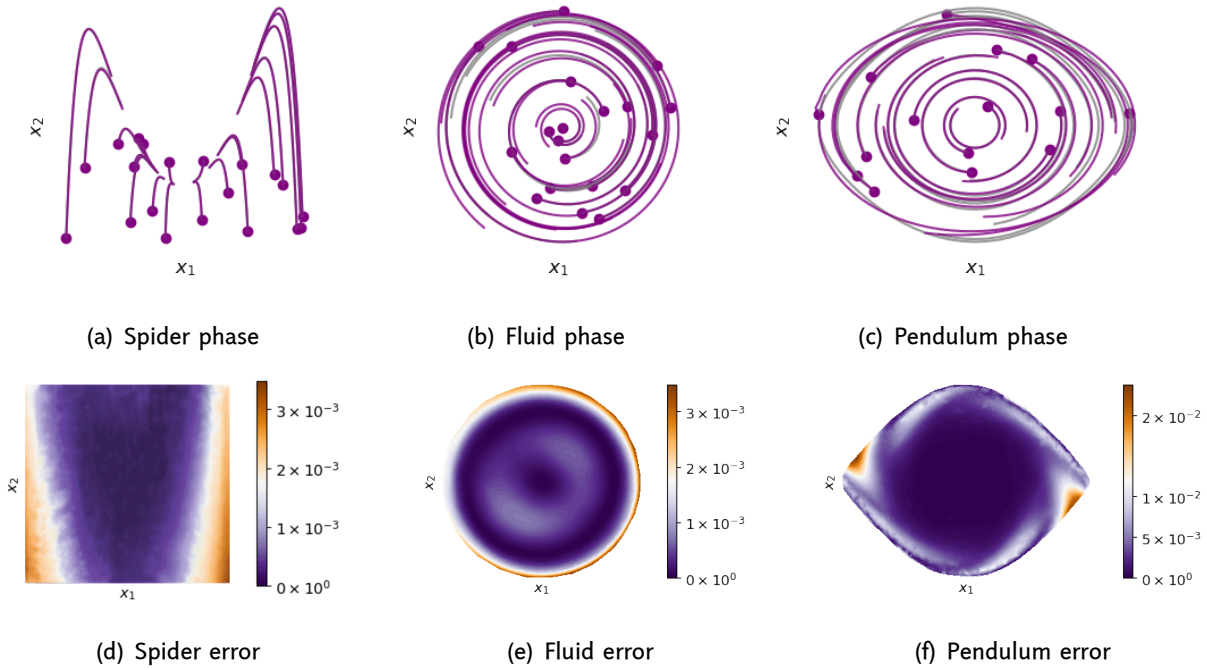
5) l_2*-regularisation loss* $\mathcal{L}_5$ which is simply the sum of the magnitude of all weights within the model, as found by [39], this technique is commonly used to prevent overfitting.

Regarding novelty, we are the first to utilise weighted-average in order to obtain smoother predictions, and the *zero loss* is also not found in literature. It is also worth noting that our loss function is not overly complex as model performance is not hugely impacted by loss components in equations *IV.8*, *IV.9* and l_2 regularisation.

We train *LREN* on three systems with analytical Koopman functions [4] discussed in section III. The detailed architecture of each system is shown in table I, we see that the 'default' hyperparameters generalise well across all systems, and our networks are all very small in size (less than 10,000 parameters). All networks are trained on a CPU (2.3 GHz Intel Core i5), using *ADAM* [40] optimiser with default momentum. Training time is around 10

	Architecture	Time Horizon	Learning Rate	α_1	α_2	α_3	α_4	α_5
Spider	$2 \times 80 \times 80 \times 1$	$5s$, $f = 0.05s^{-1}$	$1e-3$	1.	1.	$1e-5$	$1e-6$	$1e-7$
Pendulum	$2 \times 50 \times 50 \times 20$	$5s$, $f = 0.1s^{-1}$	$5e-4$	1.	1.	$1e-3$	$1e-6$	$1e-7$
Fluid	$2 \times 80 \times 80 \times 2$	$10s$, $f = 0.2s^{-1}$	$1e-3$	1.	1.	$1e-5$	$1e-6$	$1e-7$
Duffing	$2 \times 80 \times 80 \times 50$	$7.5s$, $f = 0.15s^{-1}$	$5e-4$	1.	1.	$1e-3$	$1e-6$	$1e-7$

Table I: *LREN* Hyperparameters associated with different dynamics, f is the sampling frequency



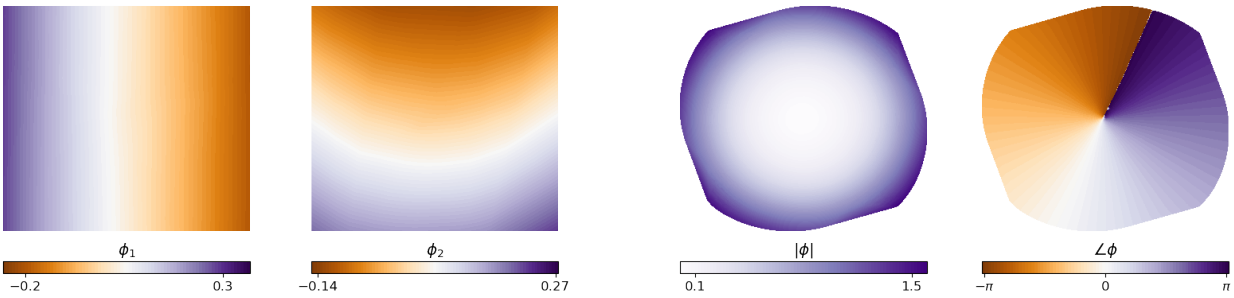
(a) Spider phase (b) Fluid phase (c) Pendulum phase

(d) Spider error (e) Fluid error (f) Pendulum error

Fig. IV.2: Validation set phase portrait and mean squared error plots for three systems considered. For each phase plot, grey lines represent ground truth, purple lines are model predictions

minutes. Figures in IV.2 show the predicted trajectories and error plots for the validation set. We observe *LREN* yields good performance for the spider and 3D fluid-flow systems, it however does relatively poorly on predicting pendulum trajectories. We attribute this to the fact that pendulum has a state-dependent Koopman spectrum, which requires a very high latent dimension to be approximated. This observation inspired our *DENIS* architecture discussed in section IV-B.

The main goal of this section is to show that *LREN*, in some cases, is able to discover the exact Koopman eigenfunctions. For each system, we perform eigen-decomposition on the



(a) (b) (c) (d)

Fig. IV.3: Eigenfunctions discovered by *LREN*. Left two: spider system, where there are two real eigenfunctions. Right two: fluid flow, where there is a pair of complex-conjugate eigenfunctions, their magnitude and phase are plotted instead

(a) Magnitude portrait (b) Phase portrait

Fig. IV.4: Eigenfunctions discovered by *pE&D*

"Koopman matrix" A, and use the eigenvectors to obtain Koopman eigenfunctions. For both spider and fluid-flow examples, we found that our eigenfunctions discovered are identical to ones found by Lusche et al. Figures in IV.3 show these eigenfunctions. In figures 3(a) and 3(b), we see the two eigenfunctions can be described equation *III.3*. In figures 3(c) and 3(d), we see the magnitude and phase of the pair of complex-conjugate eigenfunctions, which corresponds to the Hamiltonian energy of the system.

For the non-linear pendulum, we found that *LREN* is unable to identify the exact Koopman eigenfunction, this is not surprising as the pendulum is known to have a continuous Koopman eigenspectrum, which means the eigenvalues are state-dependent. We reproduce the exact eigenfunction identified by Lusche et al. by running their code which is shown

in figure IV.4.

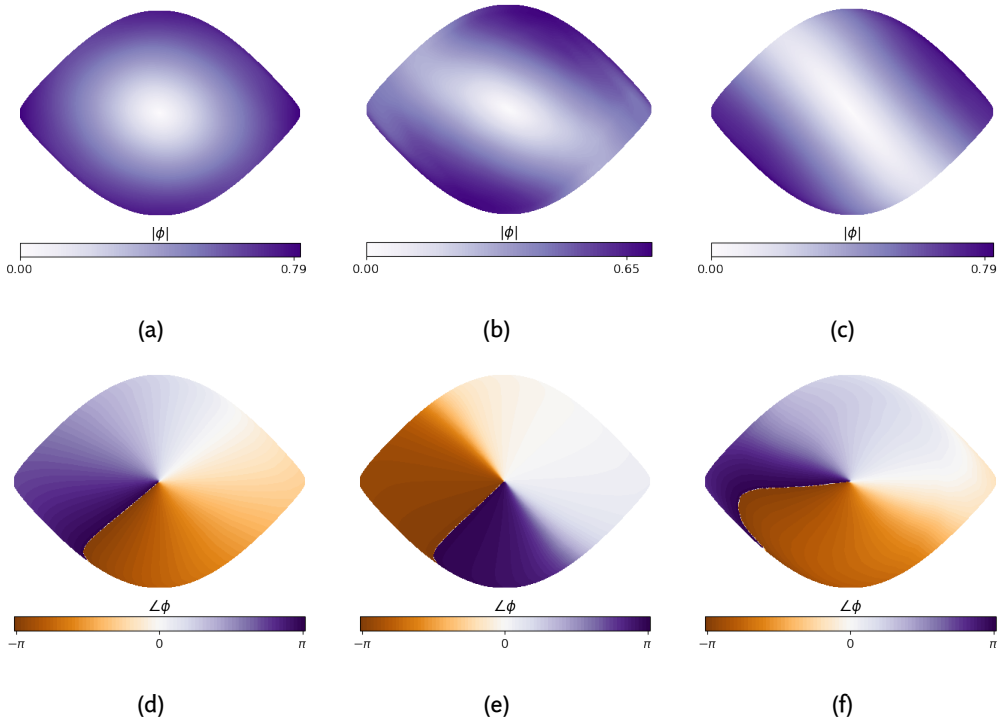


Fig. IV.5: Pendulum eigenfunctions discovered by *LREN*

We term their architecture *parameterized encoder-decoder network* (*pE&D*). Details regarding their architecture are reviewed in the next section. Figures shown in IV.5 depict eigenfunctions with three largest eigenvalues discovered by *LREN*, where we can see the close resemblance with the ground truth discovered by *pE&D*. We know that the magnitude of a pendulum's Koopman eigenfunction is simply the Hamiltonian energy, which is state-dependent; with more information encoded in its phase, which can be used to *linearly* reconstruct the state. In *LREN*, this is not possible as firstly, the observable must be linearly reconstructed, secondly, the Koopman matrix A is independent on the input. The eigenfunctions in *LREN* therefore approximate the true eigenfunction. Which explains its relative poor performance in figure 2(f).

As mentioned, all our models take roughly five to ten minutes to train on a CPU, we found this to be *significantly* faster than all existing deep-learning methods, including one by Otto et al. which utilises similar architecture. For comparison, the architecture proposed by Lusche et al. requires roughly 2 hours of GPU training for the pendulum system, and

architecture by Otto et al. requires 1 hour of CPU training. We attribute this to both the code implementation and the fact that our Koopman matrix A is learnt through standard back-propagation. Lusche et al. heavily constrain A to be in block diagonal form; Otto et al. learn A via solving a least square fitting problem during training, both are less efficient.

B. DENIS Network Architecture

In order to combat dynamics with continuous Koopman spectrum, Lusche et al utilise an auxiliary network to parameterize the Koopman operator A within each time-step. They further constrain A to be in block diagonal form, hence by definition, it is the Koopman operator. Figure 6(a) shows the unrolled dynamics of their architecture. They also utilise a non-linear decoder to reconstruct the observables. We see that under this framework, the latent vector (eigenfunctions) is *not* linearly evolved, in fact:

$$\dot{\mathbf{y}} = A(\mathbf{y})\mathbf{y} \tag{IV.10}$$

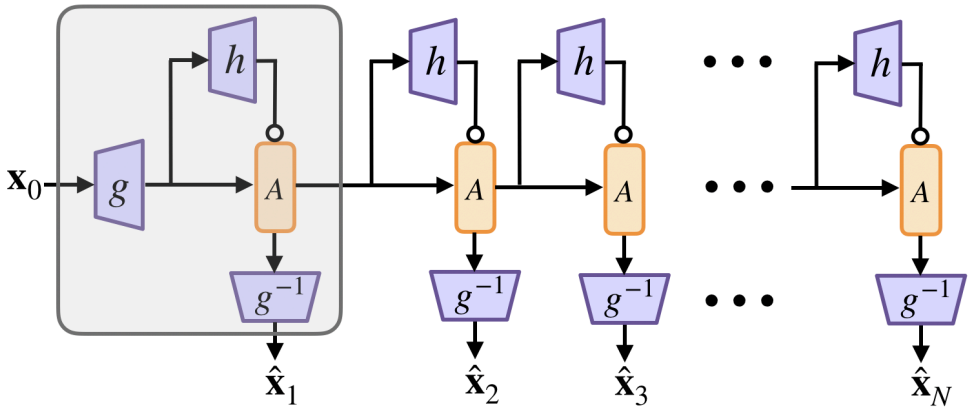
which in conjunction with the fact that $\mathbf{x}$ is not in the linear span of $\mathbf{y}$, make the system hard to optimally controll. We term their network *pE&D* where we make the following observations:

1) The auxiliary network's input does not need to be positioned after the encoder g, in fact, it may be more general to place it before g, since $h(\mathbf{y}) = h(g(\mathbf{x})) = \tilde{h}(\mathbf{x})$.
2) For systems whose Koopman eigenvalue spectrum depends *solely* on the Hamiltonian energy, for example, the non-linear pendulum, there is *no need* to parameterize the Koopman operator at each time step. Since under uncontrolled dynamics, the energy remains constant, only the initial condition is required to parameterize the Koopman operator.

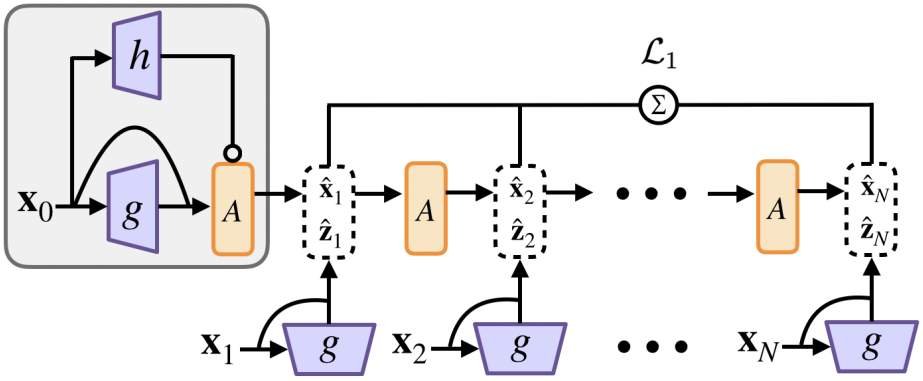
We formally write the second observation as:

$$\mathcal{A}(\mathbf{x}) = \mathcal{A}(\mathbf{x}_0) \quad \forall\, \mathbf{x} \mid \mathcal{H}(\mathbf{x}) = \mathcal{H}(\mathbf{x}_0) \tag{IV.11}$$

where $\mathcal{H}$ denotes the Hamiltonian measurement. We therefore postulate that, for systems such as the non-linear pendulum, architecture shown in figure 6(a) can be simplified by *only using* $\mathbf{x}_0$ *to parameterize* A, i.e initial-state parameterization. We empirically showed that this is true by slightly modifying *pE&D*'s model architecture, and we were still able to reproduce figures shown in IV.4.



(a) *pE&D* unrolled network architecture, note the presence of a non-linear decoder



(b) *DENIS* unrolled network architecture

Fig. IV.6: *pE&D* and *DENIS* architectures, grey box indicate the basic unit

This observation forms the basis of our novel architecture termed Deep Encoder Network with Initial State Parameterization (**DENIS**), where similar to *LREN*, we force $\mathbf{x}$ to be inside the latent space, therefore optimal control is not affected. For each trajectory, we parameterize A using an auxiliary network *once* using the initial condition. Hence the system dynamics remain completely linear for the remaining trajectory. Figure 6(b) shows the unrolled architecture of *DENIS*, compare with figure IV.1, we see that *DENIS* very slightly modifies the *LREN* architecture, however, this modification has theoretical grounds, and as we will show in section IV-D, *DENIS* indeed outperforms *LREN* in both prediction and optimal control.

C. Setup and Implementation Details

We trained *DENIS* on all dynamics presented in section III. With a batch size of 100, for 300 epochs or until convergence, whichever occurs first. Because of the auxiliary network used to parameterize the Koopman operator, the number of parameters used in *DENIS* scales quadratically with respect to the latent dimension. We did not find this to be an issue for both the training speed and RAM utilised by the CPU, since our network size is still small compared with most deep learning models. We use the same encoder architecture and hyperparameters as *LREN*, shown in table I, to help better convergence, we utilise a learning rate scheduler, specifically, a *Cosine Annealing Scheduler* [41] and a *Step Decay Scheduler* [42] are used. We found that either of these two schedulers is able to achieve a slightly lower loss than using no scheduler.

Initially, we naively attempted to visualize the Koopman eigenfunctions similar to *LREN*, we instead discovered "broken" contour plots of these eigenfunctions shown in figure IV.7. We then realized that, because we utilise the initial state to generate a different Koopman operator each time, there is no reason for *semantic consistency* across the eigenfunctions. i.e. for initial states $\mathbf{x}_{0,1}$ and $\mathbf{x}_{0,2}$, the set eigenfunctions obtained could be $\{\phi_1, \phi_2, \phi_3, \cdots, \phi_N\}$ and $\{\phi_2, \phi_N, \phi_1, \cdots, \phi_3\}$, and there is no way where we could obtain the correct way to map across these eigenfunctions. Our goal to visualize these eigenfunctions for the pendulum system is to see whether *DENIS* could discover that the Hamiltonian is one of the eigenfunctions, to do this, we need to be able to place constraints the network architecture.

Inspired by Lusche at al. we construct a network whose matrix A is in block-diagonal form. They split the eigenfunctions into real and complex conjugate pairs. In the case of complex conjugate pairs, each block B_i is generated using an auxiliary network whose outputs are μ_i and ω_i:

$$B_i = \exp(\mu_i \delta t) \begin{bmatrix} \cos \omega_i \delta t & -\sin \omega_i \delta t \\ \sin \omega_i \delta t & \cos \omega_i \delta t \end{bmatrix} \tag{IV.12}$$

The Koopman matrix therefore takes the form of:

$$A = \begin{bmatrix} B & 0 \\ 0 & C \end{bmatrix}, \quad B = \begin{bmatrix} B_1 & & \\ & \ddots & \\ & & B_n \end{bmatrix}, \quad C = \begin{bmatrix} e^{\mu_1 \delta t} & & \\ & \ddots & \\ & & e^{\mu_{N-n} \delta t} \end{bmatrix} \tag{IV.13}$$

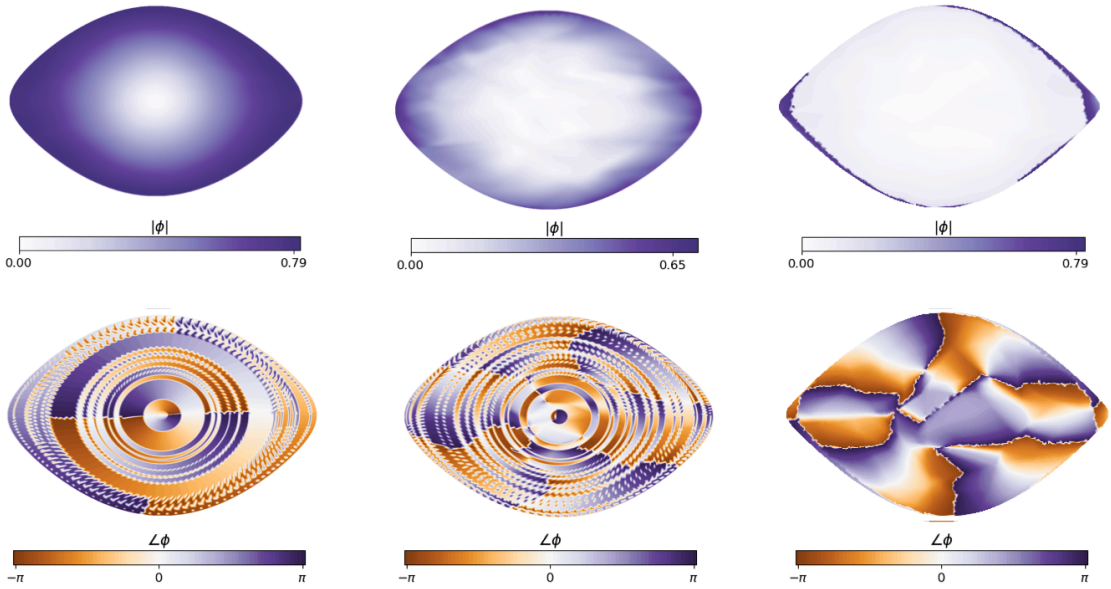


Fig. IV.7: Broke *DENIS* eigenfunctions visualization

where in the matrix above, we assumed n complex conjugate pairs, and $N - n$ real eigenfunctions. We see one potential flaw of this *pE&D* network being the number of complex and real eigenfunctions need to be assumed, and in their example, since they *knew* the pendulum system only has one complex pair of eigenfunction, they therefore explicitly set a single **B** block inside the A matrix. This is true for all systems demonstrated by Lusche et al. One other problem is that, they claim their architecture is fully interpretable, we argue that, for systems with more than one pair of complex eigenfunctions, their architecture would also run into the problem experienced by *DENIS*, i.e. there is no way of guaranteeing a consistent ordering of eigenfunctions, as shown in figure IV.7.

The benefit of adopting their Jacobian block form of A into *DENIS* is that, we can place constraint on the eigenfunctions learnt by the network. We know that there is one pair of complex eigenfunction for the pendulum, and the magnitude of which is the Hamiltonian energy, we also know the mapping from this eigenfunction to our observable is non-linear. Hence, any effort in increasing the latent dimension should attribute to a linear *approximation* of the reconstruction from the complex eigenfunction to the observables. Therefore, we form a our matrix A in block diagonal form, with *one* Jacobian block, and ensure all remaining eigenfunctions are purely real, we can postulate that the model

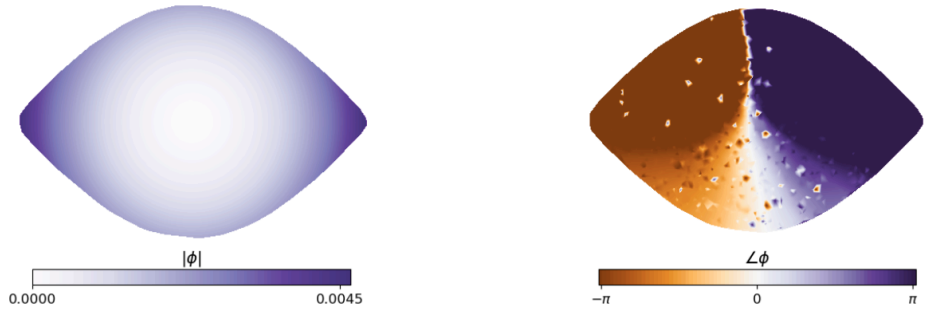


Fig. IV.8: Corrected eigenfunctions discovered by *DENIS* using Jacobian block diagonal Koopman matrix

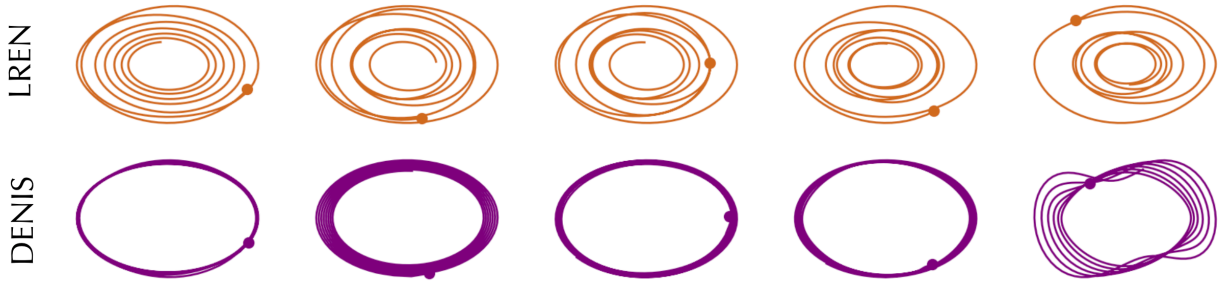


Fig. IV.9: Pendulum long time-horizon prediction for *LREN* and *DENIS*. Five random initial conditions were sampled from the validation set.

should be able learn the pair of complex eigenfunctions. Figure IV.8 shows that *DENIS* indeed is able to identify the Hamiltonian of the system, with some noise due to the linear reconstruction approximation.

D. Dynamics Reconstruction Performance Analysis

In this section, we systematically compare the performance of both *LREN* and *DENIS* on the subject of dynamics reconstruction. We will show using a variety of systems, that *DENIS* is able to:

1) Achieve better reconstruction error within the training-set prediction time horizon (maximum number of steps predicted)
2) Generalize better for prediction time horizon longer than the maximum number of time-steps trained.

Two networks are trained, *LREN* with a latent dimension of 50, and *DENIS* with a latent dimension of 30. We deliberately constrain the latent size of *DENIS* to be smaller than

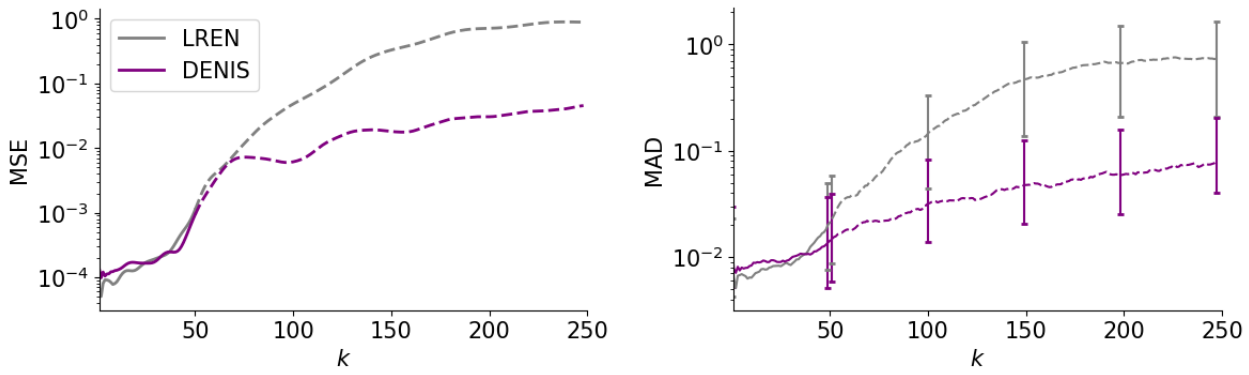


Fig. IV.10: Error vs. time-steps plot for *LREN* and *DENIS*. Plot shown for the pendulum dynamics, dashed line indicate "out of sample" performance, i.e. when prediction time steps exceeds training time horizon.

LREN and show that even under this scheme, *DENIS* out performs *LREN*. Both models are trained using parameters described in table I. For pendulum, we first visualize the predicted dynamics for a *very long time-horizon*, $T = 500$, corresponding to 50 seconds, and both models are trained only on 50 time-steps (5 seconds). Figure IV.9 shows the phase portrait of five randomly sampled results. We can clearly see that predictions for *DENIS* are well bounded around the ground truth, whereas *LREN* tend to loose energy as time increases. A more quantitative way to assess performance of the two models is through the mean squared error against time steps plot, one concern regarding utilising mean squared error is that outliers are equally accounted for. We propose to also use the *median absolute deviation* (MAD) defined as:

$$\text{MAD} = \text{median}\left(|\mathbf{x}_i - \hat{\mathbf{x}}|\right) \tag{IV.14}$$

which is the most robust [43] metric against outliers. Figure IV.10 shows that, models perform similar to each other within the training time-horizon. As expected, both models' performance degrade when tasked to predict further than training time-horizon. For both MSE and MAD metrics, *DENIS* significantly outperform *LREN* at long time horizons, showing it better approximates the underlying dynamics. We also observe that when the latent dimension of *DENIS* is the same as *LREN*, it outperforms *LREN* at lower time steps also. We repeat a similar set of experiment for the Duffing oscillator, which is a very challenging system since in the view of Koopman eigenfunctions, the two stable fixed points have identical dynamics. We see from figure IV.11 the effect of *DENIS* is more subtle, and

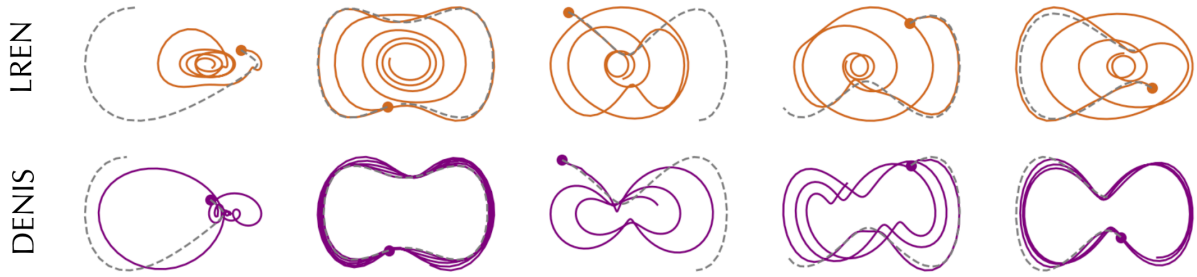


Fig. IV.11: Duffing oscillator long time-horizon prediction for *LREN* and *DENIS*. Five random initial conditions were sampled from the validation set, with ground truth shown in grey

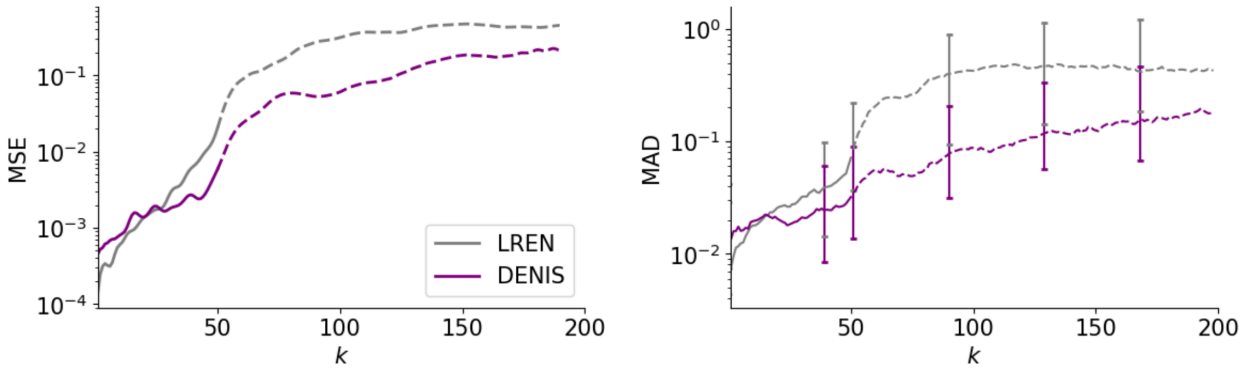
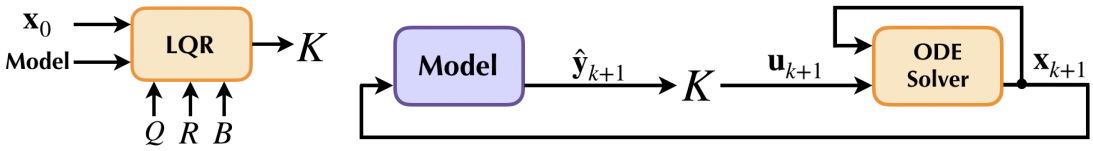


Fig. IV.12: Error vs. time-steps plot for *LREN* and *DENIS*. Plot shown for the duffing oscillator dynamics

from figure IV.11 we say that *DENIS* indeed outperforms *LREN*, this however does not stay indefinitely, in fact, their performance converge at very long time horizon (around 20 seconds). Despite this, we will shown in section IV-E that the extra performance gained by *DENIS* within these 20 seconds will mean that it outperforms *LREN* in terms of control. For other systems, we found *DENIS* achieves identical performance as *LREN*, albeit the training time required is longer due to more parameters. The identical performance can be observed by inspection, as $h(\mathbf{x})$ just needs to be a constant for these systems.

E. Koopman Optimal Control Performance Analysis

In the control-affine systems stated in section III, we assume that we know the B matrix, note in reality, this matrix may be estimated directly from data [5]. For both *LREN* and *DENIS*, because the system observables $\mathbf{x}$ are included in the latent space $\mathbf{y}$, then in



(a) Feedback gain (b) Closed-loop simulation

Fig. IV.13: Left: controller feedback gain K may be obtained through normal LQR method on the augmented system. Right: Simulation using an ODE solver (propagated using actual dynamics) to obtain actual closed-loop performance

continuous time notation:

$$\dot{\mathbf{x}} = f(\mathbf{x}) + B\mathbf{u} \tag{IV.15}$$

$$\mathbf{y} = [\mathbf{x} \quad \mathbf{z}]^T \tag{IV.16}$$

$$\dot{\mathbf{y}} = A\mathbf{y} + \tilde{B}\mathbf{u} \tag{IV.17}$$

where A is the Koopman matrix obtained through either *LREN* or *DENIS* and $\tilde{B} = [B \quad 0]^T$ is the augmented input matrix. Note in both frameworks A is discrete, it can therefore either be un-discretized or, we can instead work in discrete form, i.e. solving discrete algebraic Riccati equations. In practice, we found both methods yield similar results, for simplicity, we chose to work in continuous form for the remaining of the project. Define the quadratic cost function on the latent dimension $\mathbf{y}$ as:

$$J = \int_{t=0}^{\infty} \mathbf{y}^T \tilde{Q}\mathbf{y} + \mathbf{u}^T R\mathbf{u} \quad dt \tag{IV.18}$$

the feedback control low that minimizes the value of the cost is:

$$\mathbf{u} = -K\mathbf{x} \tag{IV.19}$$

where the feedback gain K is obtained through solving an algebraic Ricatti equation:

$$K = R^{-1}\tilde{B}^T P \tag{IV.20}$$

and P is obtained by solving the continuous time algebraic Riccati equation:

$$A^T P + PA - P\tilde{B}R^{-1}\tilde{B}^T P + \tilde{Q} = 0 \tag{IV.21}$$

Figure 13(a) shows the pictorial representation on how the controller gain matrix K may

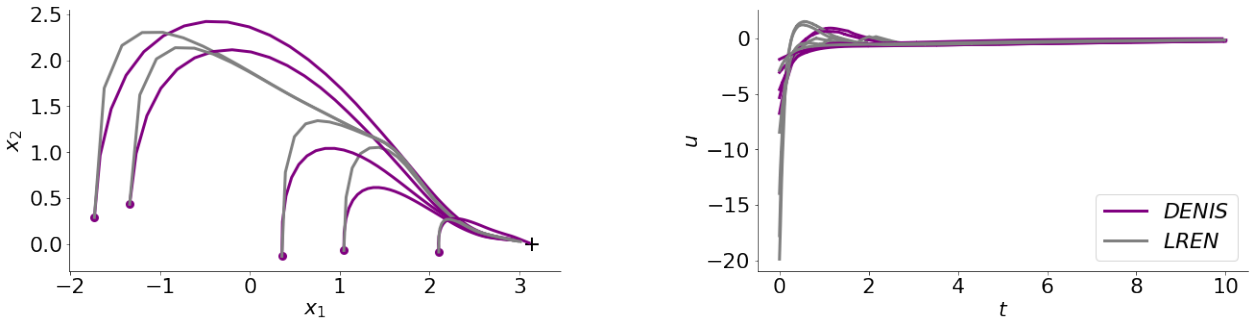


Fig. IV.14: *LREN* and *DENIS* closed-loop performance for the non-linear pendulum. Left figure shows both systems are able to approximately stabilize the system at $[\pi \quad 0]^T$, right figure shows the controller input as a function of time. Different lines of the same colour correspond to different initializations, with solid dot indicating $\mathbf{x}_0$

be obtained. In order to simulate and assess how our method performs against existing methods, we use an ordinary differential equations (ODE) solver, configured with the actual dynamics. Different models are used to predict the next latent evolution, this is then pre-multiplied with the K matrix in order to generate the next step's input to actuate the system, as shown in figure 13(b).

In order to assess different control schemes, we utilise the proposed simulation (figure 13(b)) on *LREN*, *DENIS* and iterative LQR (*iLQR*). The method used to evaluate different *closed-loop* control schemes is non-trivial. Consider the figure shown in IV.14, where we control the non-linear pendulum to the up-right position (unstable equilibrium) $\mathbf{x}_{\text{ref}} = [\pi \quad 0]^T$. From the figure we see both *LREN* and *DENIS* are able to stabilize the system at the fixed-point. It is however unclear which out-performs which, since there is a trade of between *controller input cost*: $\int_{t=0}^{\infty} \mathbf{u}^T\mathbf{u}\ dt$ and *state deviation cost*: $\int_{t=0}^{\infty} (\mathbf{x} - \mathbf{x}_{\text{ref}})^T(\mathbf{x} - \mathbf{x}_{\text{ref}})\ dt$; i.e. higher total controller input will result in faster convergence towards the desired point and vice versa. Here, we propose a *minimum energy budget* method, in order to compare different control systems in an *unbiased method*. The principal behind our method is that, for any systems, we can compare their performance if we could fix the total energy used in controller input $\int_{t=0}^{\infty} \mathbf{u}^T\mathbf{u}\ dt$ to be some constant C_u. We then can obtain how the state deviation cost changes as a function of Q. This should provide an unbiased method of comparing systems' performance. The challenge therefore, boils down to finding a controller cost R that could give rise to C_u. Note this method can be universally applied to compare between

any systems. Because our dynamics are linear in the lifted state-space, here we drive the *expected analytical* closed-loop solution to $\int_{t=0}^{\infty} \mathbf{u}^T\mathbf{u}\, dt$ for a given R.
In order to keep things general, with a slight abuse of notation, we start with the following linear system:

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} \tag{IV.22}$$

$$\mathbf{u} = K\mathbf{x} \tag{IV.23}$$

where $K = -R^{-1}B^TP$ and P is the solution to the algeneric Riccati equation:

$$A^TP + PA - PBR^{-1}B^TP + Q = 0 \tag{IV.24}$$

the closed-loop dynamics can be written as:

$$\mathbf{x}(t) = e^{t(A+BK)}\mathbf{x}_0 \tag{IV.25}$$

note that the total cost is the trace of the solution to the Riccati equation:

$$C = \int_0^{\infty} \mathbf{x}^TQ\mathbf{x} + \mathbf{u}^TR\mathbf{u} = \mathsf{Tr}(P) \tag{IV.26}$$

the controller cost can therefore be written as:

$$\int_0^{\infty} \mathbf{u}^T\mathbf{u}\, dt = R^{-1}\left(\mathsf{Tr}(P) - \int_0^{\infty} \mathbf{x}^TQ\mathbf{x}\, dt\right) \tag{IV.27}$$

expanding out $\int_0^{\infty} \mathbf{x}^TQ\mathbf{x}$:

$$\begin{aligned}\int_0^{\infty} \mathbf{x}^TQ\mathbf{x}\, dt &= \int_0^{\infty} \mathsf{Tr}\left(Q\mathbf{x}\mathbf{x}^T\right)\, dt \\ &= \mathsf{Tr}\left(\int_0^{\infty} e^{t(A+BK)}Q\mathbf{x}_0\mathbf{x}_0^Te^{t(A+BK)^T}dt\right)\end{aligned} \tag{IV.28}$$

As we average over all initial conditions, which are uniformly sampled across the state space, $\langle \mathbf{x}_0\mathbf{x}_0^T = I\rangle$:

$$\int_0^{\infty} \left\langle \mathbf{x}^TQ\mathbf{x}\right\rangle_{\mathbf{x}_0}\, dt = \mathsf{Tr}\left(\int_0^{\infty} e^{t(A+BK)}Qe^{t(A+BK)^T}dt\right) = \mathsf{Tr}(M) \tag{IV.29}$$

where M is the solution to the Lyapunov equation:

$$(A+BK)M + M(A+BK)^T + Q = 0 \tag{IV.30}$$

Finally, substituting in equation *IV.27*, we arrive at the following theorem for our *energy budget* calculation:

Theorem 1: The expected total controller input across all initialization is:

$$\int_0^\infty \left\langle \mathbf{u}^T \mathbf{u} \right\rangle_{x_0} dt = R^{-1} \,\mathrm{Tr}(P - M) \tag{IV.31}$$

where M is the solution through the Lyapunov equation:

$$(A + BK)M + M(A + BK)^T + Q = 0 \tag{IV.32}$$

The following algorithm utilises this theorem in to obtain an R that corresponds to the energy budget C_u, then this R is used to obtain the gain matrix K. We then simulate the system with a range of Q (state cost matrix), and plot total state cost $\int_0^\infty x^T x dt$. We repeat this process many times, using a variety of fixed points in order to obtain a state cost mean as a function of Q. This function can be visualized for a variety of control schemes. Algorithm shown in 1 shows the detailed pseudocode for this evaluation method.

We compare both *LREN* and *DENIS* against a local linearization approach called *iterative LQR* [44], which iteratively approximates solution by solving linearized versions of the problem via LQR. We abbreviate this as *iLQR*. Note one obvious advantage between any Koopman approach over *iLQR* is that, system's dynamical equations are required for control to be possible, whereas in ours and others Koopman approaches, we assume the system dynamics is inaccessible. Nevertheless, this comparison makes sense as we wish to demonstrate the power of the Koopman framework. We found an off-the-shelf *iLQR* library[4], and modified it accordingly to control our dynamical systems. Figures 15(a) 15(c) show that for the pendulum and duffing systems, utilising energy budget evaluation framework, *DENIS* consistently outperforms *LREN*. We attribute this to the better prediction performance that *DENIS* achieves. *DENIS*'s control performance is also comparable with *iLQR*, and slightly outperforms it occasionally. The uncertainty bar is calculated using multiple initial system states.

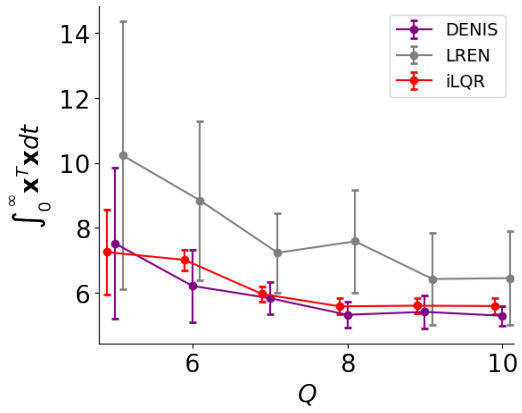
The Koopman model predictive control *MPC* is also a popular framework used for controller non-linear dynamics via the extended states. It is more flexible than *LQR* as it allows more constraints to be placed on controlling the system. We show in the appendix how the Koopman framework applies to *MPC*, our work on this is almost completely based on Mezic's [15] work. The Koopman *MPC* was implemented from scratch. Figures 15(b) and 15(d)

[4]project location: https://github.com/anassinator/ilqr

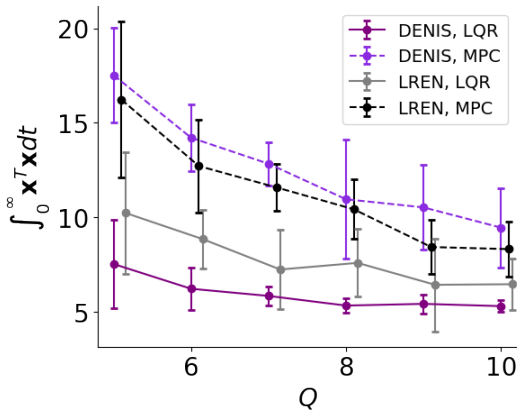
Algorithm 1 Control Evaluation

procedure control_eval(model, $\{\mathbf{x}_0\}, \tilde{B}, \tilde{Q}, C_u$) ▷ trained model, list of initial conditions, padded input matrix and state cost, desired energy budget
  $c \leftarrow 0$ ▷ Initialize C_u
  $R \leftarrow 1$ ▷ Initial guess for R
  $\langle C_x \rangle \leftarrow 0$ ▷ Initialize average state cost
  for $\mathbf{x}_0$ in $\{\mathbf{x}_0\}$ **do**
    $A \leftarrow \text{model}(\mathbf{x}_0)$ ▷ obtain Koopman operator from the model
    while $c \neq C_u$ **do** ▷ bisection search R
      $P \leftarrow$ Riccati Solver$(A, \tilde{B}, \tilde{Q}, R)$ ▷ solve equation *IV.21*
      $M \leftarrow$ Lyapunov Solver$(A, \tilde{B})$ ▷ solve equation *IV.30*
      $c \leftarrow R^{-1}\,\text{Tr}(P - M)$
      $R \leftarrow \text{bisect}(R, c)$ ▷ update new R according to the current energy used
    end while
    $K \leftarrow R^{-1}\tilde{B}^T P$ ▷ obtain feedback gain using the resulting R
    $[\mathbf{x}_1, \cdots, \mathbf{x}_T] \leftarrow$ ODE Solve$(\text{model}, \mathbf{x}_0, K)$ ▷ simulate closed-loop dynamics
    $C_x \leftarrow \text{Integrate}([\mathbf{x}_1, \cdots, \mathbf{x}_T])$ ▷ obtain total state cost through numerical integration
    $\langle C_x \rangle \leftarrow \text{update}(C_x)$ ▷ update average state cost
  end for
  return $\langle C_x \rangle$
end procedure

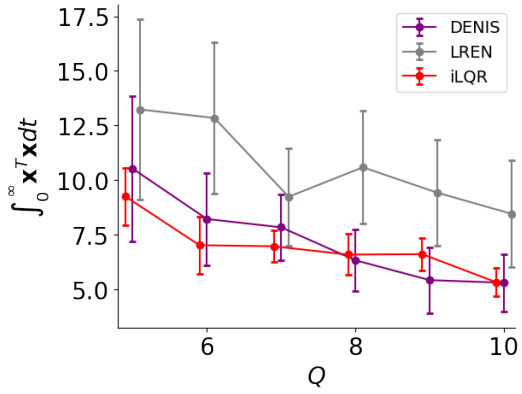
show that, *MPC* is consistently out-performed by *LQR* on the extended state-space. This is not surprising as *LQR* solves a problem with infinite-time horizon, whereas for *MPC* we specified a finite time horizon of 2 seconds. It is interesting that *DENIS* is outperformed by *LREN* under *MPC*, although with a closer look of figure IV.12, we see that *DENIS*'s prediction performance is slightly worse than *LREN*'s for the first few (20) time-steps. Note we could have repeated the experiment on *MPC* by increasing the prediction time horizon, however, as our implementation was not very efficient (MPCs are normally done using hardware acceleration [45]), we found solving a high dimensional convex optimisation problem to be



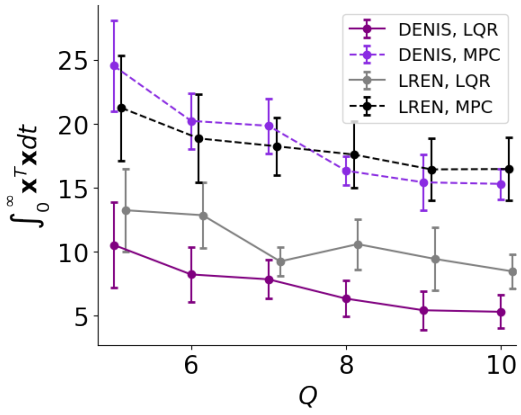
(a) Pendulum: *DENIS* vs. *LREN* vs. iLQR



(b) Pendulum: MPC vs. LQR



(c) Duffing: *DENIS* vs. *LREN* vs. iLQR



(d) Duffing: MPC vs. LQR

Fig. IV.15: Control performance comparison using the energy budget method. Non-linear pendulum and duffing oscillator are used where models are tasked to drive the system $[\pi\ 0]^T$ and $[0\ 0]$ respectively. Left: comparison between our two DL based approaches and iterative LQR. Right: comparison between Koopman model predictive control (MPC) and controlling using LQR on the expanded state-space. In all cases, we require a energy budget of $C_u = 100$

too time consuming. The time complexity is worsened by the need to complete the energy budget evaluation.

One other clarification is that, the energy budget algorithm shown in 1 does not apply to *iLQR* and MPC. Instead, we directly search for C_u for these two methods via simulations. *Scipy*'s optimisation library is used, this means figures shown in IV.15 take a significant amount of time to generate.

F. Implementation Details and Ablation Studies

In this section, we perform ablation study on some key aspects of the *DENIS* network. Specifically, we investigate how the following attributes effect model performance:

- Using radial basis transformation before the first encoder layer
- The size of the latent dimension
- Depth of the model
- Prediction time horizon during training
- Effect of including "zero cost" in the loss function, i.e. ensuring $g(0) = 0$

1) RBF Transformation: First, we investigate whether radial basis transformation before the first encoder layer helps with model prediction performance. The motivation behind this is to increase the capacity of the network in a manner that does not drastically increase the number of model parameters (recall number of parameters in *DENIS* scales with dN^2 where N is the size of the latent dimension and d is the depth of the network). Radial basis function kernel, or RBF kernel, is a popular kernel function used in various kernelized learning algorithms [46], where non-liearities are added in an efficient manner. Since we need to lift our state-space dimension anyway, we could instead add an radial basis transformation before the first encoder layer, i.e.:

$$\begin{bmatrix} \hat{\mathbf{x}}_1 \\ \vdots \\ \hat{\mathbf{x}}_n \end{bmatrix} = \begin{bmatrix} \phi\left(\sigma_1 \left\|\boldsymbol{x}_1 - \boldsymbol{c}_1\right\|_2\right) & \cdots & \phi\left(\sigma_m \left\|\boldsymbol{x}_1 - \boldsymbol{c}_m\right\|_2\right) \\ & \vdots & \\ \phi\left(\sigma_1 \left\|\boldsymbol{x}_n - \boldsymbol{c}_1\right\|_2\right) & \cdots & \phi\left(\sigma_m \left\|\boldsymbol{x}_n - \boldsymbol{c}_m\right\|_2\right) \end{bmatrix} \tag{IV.33}$$

where ϕ denotes the kernel function (we choose the standard Gaussian kernel), and $\sigma_i, \mathbf{c}_i$ are learnt length scales and centres. By construction of *DENIS*, both the encoder and the auxiliary network could receive an RBF layer as its first layer. We also experimented with having RBF at every single layer, however, this is computationally expensive and yielded poor results. Figures in IV.16 shows the prediction performance analysis for both the pendulum and duffing oscillator. We omit showing MAD measurement in the interest of space, as the measurement exhibits identical behaviour as MSE measurement. Figures 16(a) and 16(c) shows the performance of models with or without RBF transformation on both/neither/either/one of the encoder/auxiliary networks (architectures labeled as 1.1 and 1.2 in appendix table II). We see results are inconclusive. We also compare RBF transformation on both encoder and auxiliary networks against a model with a much higher

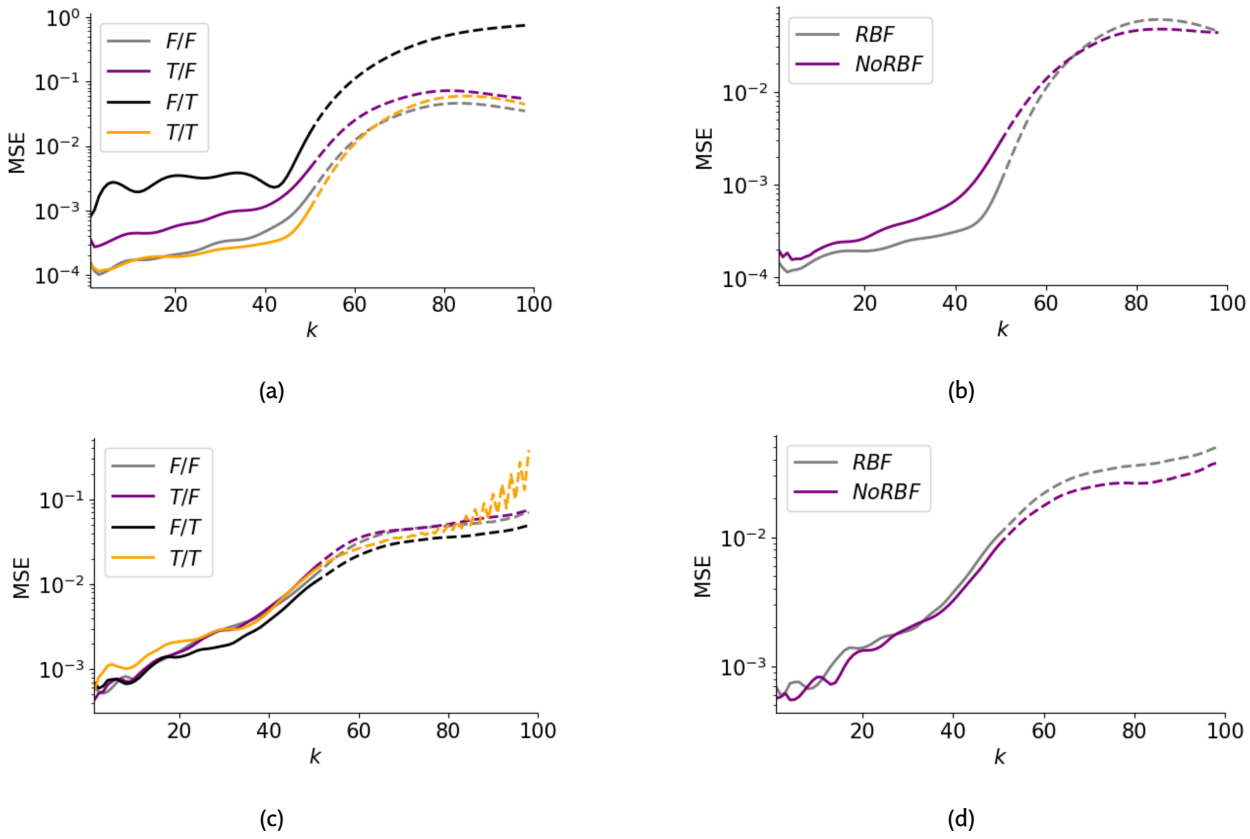
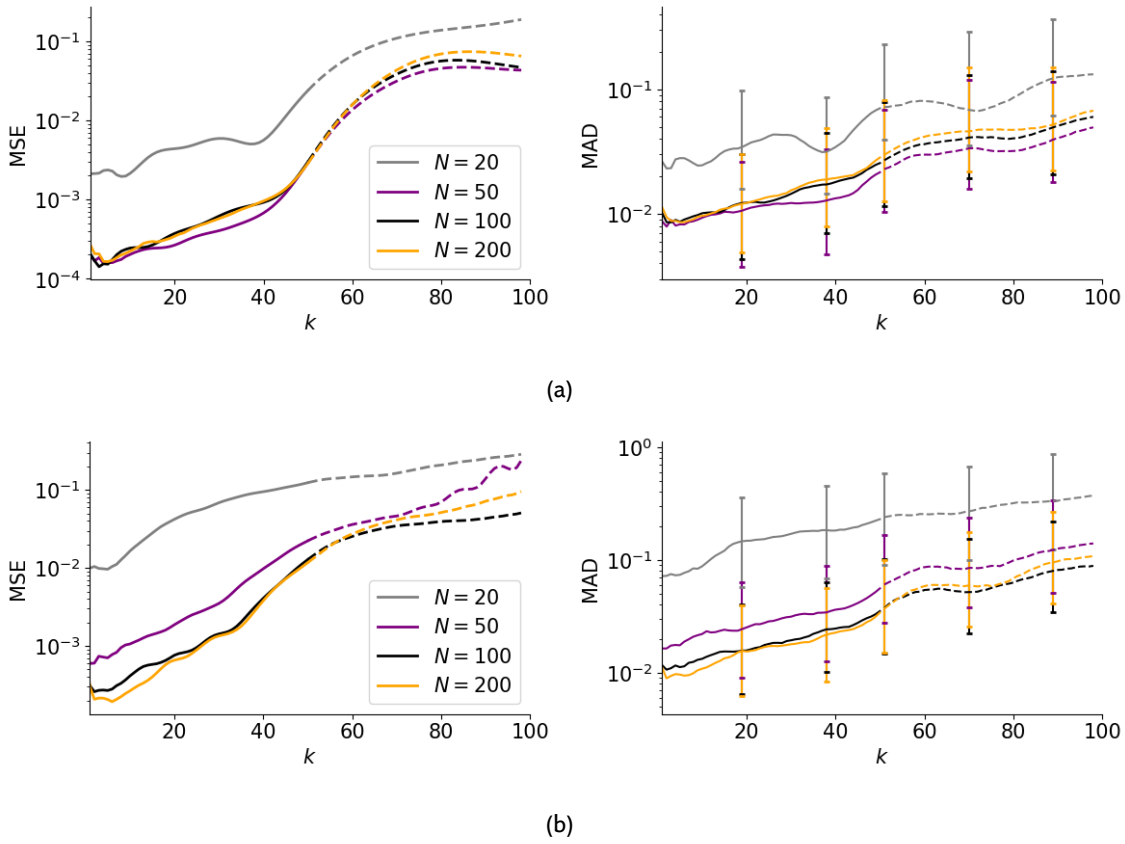


Fig. IV.16: Effect of RBF transform. Top row: results for the pendulum system, bottom row: results for duffing oscillator. Left column: investigation on whether to apply a RBF layer on the encoder and/or auxiliary network (True/False for encoder/auxiliary network respectively). Right column: comparison between the best-performing combination in the top plot, and a network with much higher depth

depth (hence more non-linearity, architecture is labeled as 1.3 in appendix table II), results seen in figures 16(b) and 16(d) are also inconclusive. Hence we postulate that in the case of *DENIS*, adding RBF transformation neither hurts or benefits model prediction performance.

2) Size of Latent Dimension: We then investigate how the size of the latent dimension effect model performance. Intuitively, model performance should only saturate with increasing depth, since we can always have repeated Koopman eigenfunctions if there are redundancies. We investigate a set of architectures with latent dimensions being $N \in [20, 50, 100, 200]$ with detailed architectures described in table II (experiment labels 2.1 to 2.4 inclusive). Figures shown in IV.17 demonstrate that, for both pendulum and duffing



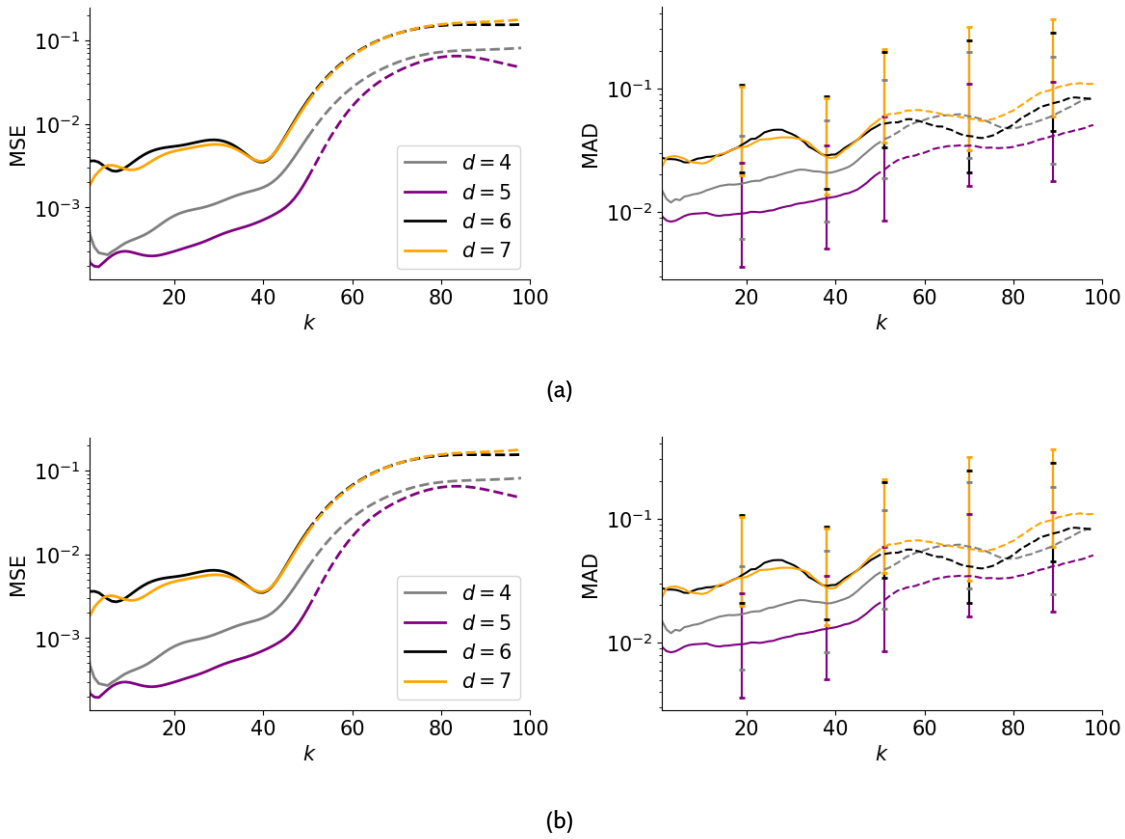
(a)

(b)

Fig. IV.17: Effect of latent dimension N on model performance. Top row: results for the pendulum system, bottom row: results for duffing oscillator

oscillator, increasing latent dimension indeed can significantly aid model performance, but only up to a certain extent. This does confirm with our intuition, however it does demonstrate that there seems to be a bound on model performance under our Koopman approximation scheme. In fact, as shown in figure 17(a), over parameterizing the Koopman operator actually will slightly hurt model performance, possibly due to the fact that it is harder to train, and can easily be stuck at local minimums.

3) Network Depth: The effect of network depth is also investigated. In theory, the greater depth, the greater the representational capacity *DENIS* has, hence intuitively, increasing network depth should not decrease model performance. However, experiment shows this may not generally be the case. From figure 18(a), we see that at high depth, performance on the pendulum actually decreases, whereas figure 18(b) agrees our intuition. Training deeper networks has always been a challenge in the DL community, we refer to popular frameworks



(a)

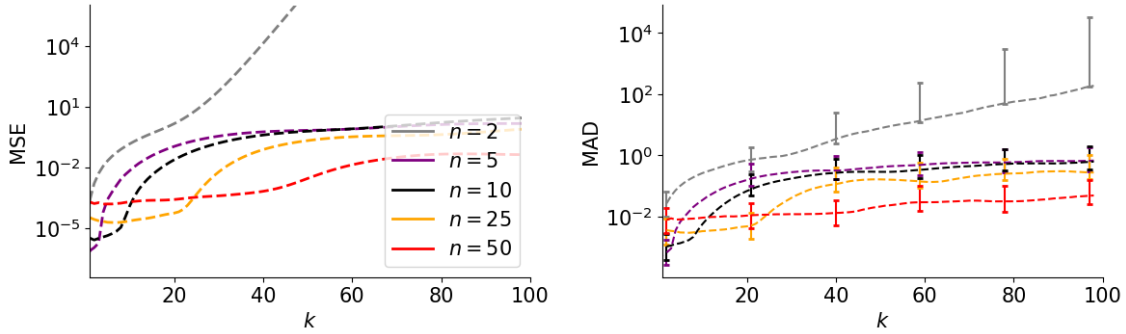
(b)

Fig. IV.18: Effect of network depth on model performance. Top row: results for the pendulum system, bottom row: results for duffing oscillator

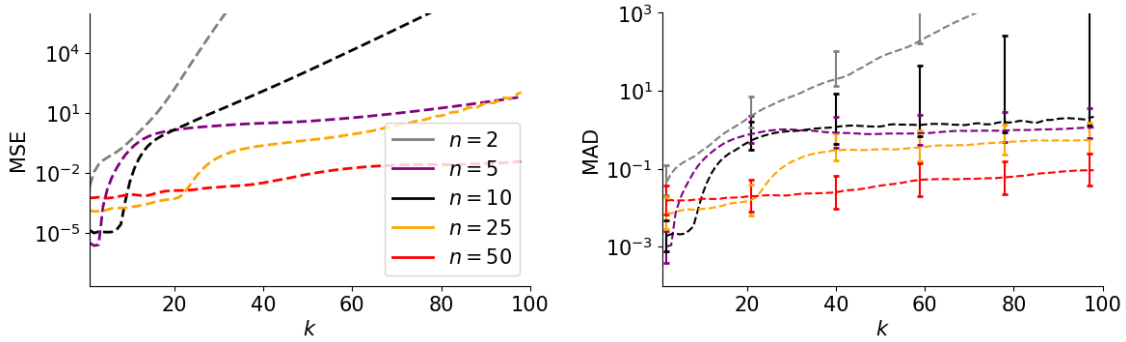
such as ResNet [47] to combat the issue of vanishing gradient for future work. For now, we caution against naively increase model complexity, since increases both training time, and may even decrease model performance. Detailed architectures can be found in table II labeled as 3.1-3.4.

4) Number of Time Shifts: It is computationally expensive to train the network when we linearly step-through the network using the Koopman matrix A multiple times. We show in this section that it is essential for at least a few time-steps to be evolved in order to achieve good control performance. From figure IV.19 we see how models trained on insufficient number of time-steps behave poorly in terms of long time-horizon prediction. This is quantitatively shown in figures 20(a) and 20(b). Where we see that in particular, when $n = 2$, the dynamic learnt is highly unstable, hence we anticipate the control performance using that model will be particularly poor.

Fig. IV.19: Phase plot of duffing oscillator validation result, from left to right: prediction time horizon during training $n = 5, 10, 25, 50$, solid purple lines indicate trained trajectories' prediction, dashed purple lines indicate "out of sample" predictions



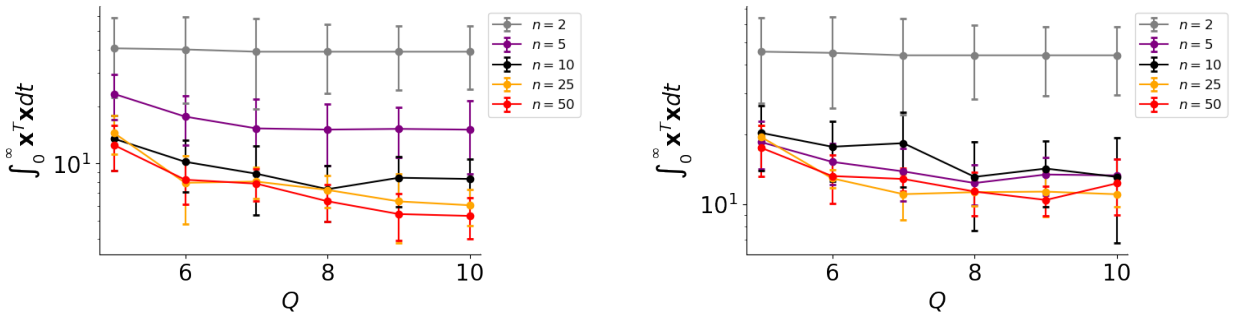
(a)



(b)

Fig. IV.20: Effect of number of time shifts (prediction time horizon during training). Top row: results for the pendulum system, bottom row: results for duffing oscillator

In order to investigate how control performance is affected by the number of time-shifts, we employ the *energy budget* described in section IV-E. Figures in IV.21 confirm that, the control performance is especially poor when the prediction time-horizon is only 2 time-steps. This confirms the poor control-performance exhibited by algorithms such as eDMD



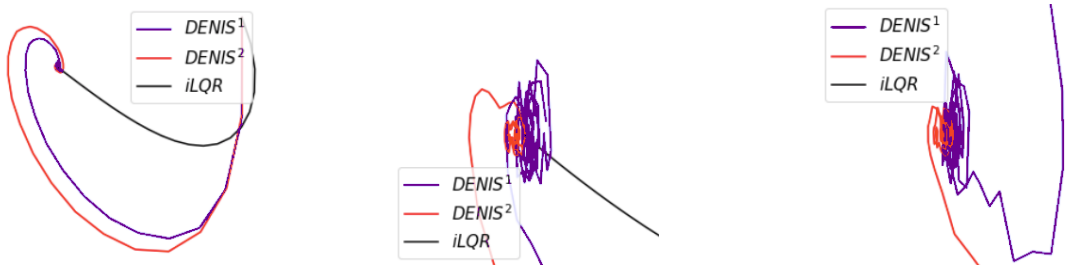
(a) (b)

Fig. IV.21: Control performance under number of time shifts (prediction time horizon during training). Top row: results for the pendulum system, bottom row: results for duffing oscillator

(as discussed in section I-C), since our network becomes a regularised eDMD when $n = 2$. As the number of shifts during training increases, the extra control performance decreases, as anticipated, because of the feedback loop. Interestingly, for the duffing oscillator, we see that performance at $n = 5$ is on average higher than $n = 10$, we attribute this to the behaviour observed in figure 20(b), where under the mean squared error measurement, we see that $n = 5$ out performs $n = 10$, meaning the network is less stable at $n = 10$. Since our measurement for control performance takes outliers into account, we anticipated worse performance at $n = 10$.

Therefore we can conclude, including time-shifts in fitting the Koopman matrix A is indeed beneficial to control performance. This is true for both *DENIS*, *LREN* and should also be true for other frameworks.

5) Zero Constraint Loss: As mentioned in section IV-A, we utilize an extra loss component to enforce $g(0) = 0$ during training. To observe the effect of this on the control performance, we utilize two networks trained with and without $\mathcal{L}_4$ (detailed architecture shown in table II experiments 5.1 and 5.2). Figures in IV.22 show that when the network achieves $g(0) = 0$, it is more stable around the fixed-point.



(a) (b) (c)

Fig. IV.22: Controlled dynamics for *DENIS* trained with and without $g(0) = 0$ constraint. Right most figure depicts the zoomed out trajectories, remaining figures are zoomed in around $[0 \quad 0]$

V. *DEINA*: Double Encoder for Input Non-Affine Systems

VI. Conclusion

VII. Appendix

A. Koopman Model Predictive Control

In this section, we summarize and extend the work done by Mezic et al. On the Koopman model predictive control. We implement a version of this using CVXPY.

Consider a discrete-time nonlinear controlled dynamical system

$$x^+ = f(x, u), \tag{VII.1}$$

where $x \in \mathbb{R}^n$ and $u \in \mathcal{U} \subset \mathbb{R}^m$. Following the Koopman framework, this system can be lifted into a linear regime where

$$\begin{aligned} z^+ &= Az + Bu \\ \hat{x} &= Cz \end{aligned} \tag{VII.2}$$

where $z \in \mathbb{R}^N$ and $\hat{x}$ is the prediction of x^+. $B \in \mathbb{R}^{N \times m}$ and $C \in \mathbb{R}^{n \times N}$. In the case of *LREN* and *DENIS*, C is just the first n rows of A. Initial condition is given by

$$z_0 = \boldsymbol{\psi}(x_0) := \begin{bmatrix} \psi_1(x_0) \\ \vdots \\ \psi_N(x_0) \end{bmatrix} \tag{VII.3}$$

where $\psi_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, N$ are the (typically) nonlinear lifting functions.

Model predictive control on the lifted state solves at each time instance k of the closed-loop operator the optimization problem:

$$\begin{aligned} \underset{u_i, z_i}{\text{minimize}} \quad & J\left((u_i)_{i=0}^{N_p-1}, (z_i)_{i=0}^{N_p}\right) \\ \text{subject to} \quad & z_{i+1} = Az_i + Bu_i, \quad i = 0, \ldots, N_p - 1 \\ & E_i z_i + F_i u_i \leq b_i, \quad i = 0, \ldots, N_p - 1 \\ & E_{N_p} z_{N_p} \leq b_{N_p} \\ \text{parameter} \quad & z_0 = \boldsymbol{\psi}(x_k) \end{aligned} \tag{VII.4}$$

where N_p is the prediction horizon, matrices $E_i \in \mathbb{R}^{n_c \times N}$, $F_i \in \mathbb{R}^{n_c \times m}$ and vector $b_i \in \mathbb{R}^{n_c}$ define state and input polyhedral constraints. The quadratic cost function J is given by

$$\begin{aligned} J\left((u_i)_{i=0}^{N_p-1}, (z_i)_{i=0}^{N_p}\right) &= z_{N_p}^\top Q_{N_p} z_{N_p} + q_{N_p}^\top z_{N_p} \\ &+ \sum_{i=0}^{N_p-1} z_i^\top Q_i z_i + u_i^\top R_i u_i + q_i^\top z_i + r_i^\top u_i \end{aligned} \tag{VII.5}$$

where $q_i \in \mathbb{R}^N$ places cost on the lifted state (i.e. penalizing nonlinear functions of the original state x). The closed-loop operation of the lifting-based MPC can be summarized by the following algorithm, where $U_{1:m}^\star$ denotes the first m components of U^*:

Algorithm 2 Koopman MPC

1: **for** $k = 0, 1, \ldots$ **do**
2: Set $z_0 := \boldsymbol{\psi}(x_k)$
3: Solve to get an optimal solution U^*
4: Set $u_k = U_{1:m}^\star$
5: $x_{k+1} = f(x_k, u_k)$ ▷ Solve using ODE45 to next timestep
6: **end for**

B. Detailed DENIS Architecture

	system	enc_shape	aux_shape	n_shifts	enc_rbf	aux_rbf	$\{\alpha_1, \cdots, \alpha_5\}$	lr	lr_sch_step	lr_sch_gamma	epochs
1.1	pendulum	$2 \times 50 \times 25 \times 25 \times 50$	$2 \times 50 \times 25 \times 25 \times 50$	51	T/F	T/F	$1,\ 1,\ 1e-3,\ 1,\ 1e-7$	$5e-4$	50	0.5	300
1.2	duffing	$2 \times 50 \times 25 \times 25 \times 50$	$2 \times 50 \times 25 \times 25 \times 52$	51	T/F	T/F	$1,\ 1,\ 1e-3,\ 1,\ 1e-7$	$5e-4$	50	0.5	500
1.3	pendulum/duffing	$2 \times 25 \times 25 \times 25 \times 25 \times 25 \times 50$	$2 \times 25 \times 25 \times 25 \times 25 \times 25 \times 52$	51	F	F	$1,\ 1,\ 1e-3,\ 1,\ 1e-7$	$5e-4$	50	0.5	500
2.1	pendulum/duffing	$2 \times 5 \times 10 \times 10 \times 20$	$2 \times 5 \times 10 \times 10 \times 22$	51	F	F	$1,\ 1,\ 1e-3,\ 1,\ 1e-7$	$5e-4$	25	0.5	200
2.2	pendulum/duffing	$2 \times 5 \times 25 \times 25 \times 50$	$2 \times 5 \times 25 \times 25 \times 52$	51	F	F	$1,\ 1,\ 1e-3,\ 1,\ 1e-7$	$5e-4$	50	0.5	300
2.3	pendulum/duffing	$2 \times 25 \times 50 \times 50 \times 100$	$2 \times 25 \times 50 \times 50 \times 102$	51	F	F	$1,\ 1,\ 1e-3,\ 1,\ 1e-7$	$1e-3$	100	0.5	500
2.4	pendulum/duffing	$2 \times 50 \times 100 \times 100 \times 200$	$2 \times 50 \times 100 \times 100 \times 202$	51	F	F	$1,\ 1,\ 1e-3,\ 1,\ 1e-7$	$1e-3$	100	0.5	500
3.1	pendulum/duffing	$2 \times 25 \times 25 \times 50$	$2 \times 25 \times 25 \times 52$	51	F	F	$2,\ 1,\ 1e-3,\ 1,\ 1e-7$	$1e-4$	25	0.5	200
3.2	pendulum/duffing	$2 \times 25 \times 25 \times 25 \times 50$	$2 \times 25 \times 25 \times 25 \times 52$	51	F	F	$2,\ 1,\ 1e-3,\ 1,\ 1e-7$	$5e-4$	50	0.5	300
3.3	pendulum/duffing	$2 \times 25 \times 25 \times 25 \times 25 \times 50$	$2 \times 25 \times 25 \times 25 \times 25 \times 52$	51	F	F	$1,\ 1,\ 1e-3,\ 1,\ 1e-5$	$5e-4$	50	0.5	500
3.4	pendulum/duffing	$2 \times 25 \times 25 \times 25 \times 25 \times 25 \times 50$	$2 \times 25 \times 25 \times 25 \times 25 \times 25 \times 52$	51	F	F	$1,\ 1,\ 1e-3,\ 1,\ 1e-5$	$5e-4$	100	0.5	500
4.1	pendulum	$2 \times 10 \times 25 \times 25 \times 50$	$2 \times 10 \times 25 \times 25 \times 52$	2 -51	F	F	$1,\ 1,\ 1e-3,\ 1,\ 1e-7$	$5e-4$	50	0.5	300
4.2	duffing	$2 \times 10 \times 25 \times 25 \times 50$	$2 \times 10 \times 25 \times 25 \times 52$	2 -51	F	F	$1,\ 1,\ 1e-3,\ 1,\ 1e-7$	$1e-4$	100	0.5	500
5.1	duffing	$2 \times 10 \times 25 \times 25 \times 50$	$2 \times 10 \times 25 \times 25 \times 52$	51	F	F	$1,\ 1,\ 1e-3,\ 2,\ 1e-7$	$1e-4$	100	0.5	500
5.2	duffing	$2 \times 10 \times 25 \times 25 \times 50$	$2 \times 10 \times 25 \times 25 \times 52$	51	F	F	$1,\ 1,\ 1e-3,\ 0,\ 1e-7$	$1e-4$	100	0.5	500

Table II: *DENIS* detailed architecture for different experiments done during ablation. **enc_shape** and *aux_shape* are the encoder and auxiliary networks' shapes respectively. **n_shifts** denotes the number of times the linear dynamic is evolved, i.e. prediction time horizon. **enc_rbf** and **aux_rbf** denote whether the encoder and/or the auxiliary network utilizes radial basis transformation for their first layers. The set of αs are the hyper-parameters associated with each loss function described in section IV-A. **lr** denotes learning rate, **lr_sch_step** and **lr_sch_gamma** denote the decreasing learning rate scheduler's parameters (number of iterations per decay and decay each time)

References

[1] B. Lantos and L. Márton, *Nonlinear Control of Vehicles and Robots*, ser. Advances in Industrial Control. London: Springer London, 2011.

[2] F. Corinto and A. Torcini, *Nonlinear Dynamics in Computational Neuroscience*. 2019.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, ser. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016.

[4] B. Lusch, J. N. Kutz, and S. L. Brunton, "Deep learning for universal linear embeddings of nonlinear dynamics," *Nature Communications*, vol. 9, no. 1, p. 4950, Dec. 2018.

[5] S. E. Otto and C. W. Rowley, "Linearly Recurrent Autoencoder Networks for Learning Dynamics," *SIAM Journal on Applied Dynamical Systems*, vol. 18, no. 1, pp. 558–593, Jan. 2019.

[6] *Nonlinear dynamics in biological systems*. New York, NY: Springer Berlin Heidelberg, 2016.

[7] G. Buzzi-Ferraris and F. Manenti, *Nonlinear systems and optimization for the chemical engineer: solving numerical problems*. Weinheim: Wiley-VCH, 2014.

[8] R. Dieci and C. Chiarella, Eds., *Nonlinear economic dynamics and financial modelling: essays in honour of Carl Chiarella*. Cham: Springer, 2014.

[9] M. Kamb, E. Kaiser, S. L. Brunton, and J. N. Kutz, "Time-Delay Observables for Koopman: Theory and Applications," Feb. 2020.

[10] B. O. Koopman, "Hamiltonian Systems and Transformation in Hilbert Space," *Proceedings of the National Academy of Sciences*, vol. 17, no. 5, pp. 315–318, May 1931.

[11] M. Korda and I. Mezic, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149–160,

[12] ——, "Optimal construction of Koopman eigenfunctions for prediction and control," *IEEE Transactions on Automatic Control*, 2020.

[13] I. Abraham and T. D. Murphey, "Active Learning of Dynamics for Data-Driven Control Using Koopman Operators," *IEEE Transactions on Robotics*, vol. 35, no. 5, pp. 1071–1083, Oct. 2019.

[14] J. Willems, "Paradigms and puzzles in the theory of dynamical systems," *IEEE Transactions on Automatic Control*, vol. 36, no. 3, pp. 259–294, Mar. 1991.

[15] M. Korda and I. Mezic, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149–160, Jul. 2018.

[16] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Generalizing Koopman Theory to allow for inputs and control," *arXiv:1602.07647 [math]*, Feb. 2016.

[17] M. O. Williams, M. S. Hemati, S. T. Dawson, I. G. Kevrekidis, and C. W. Rowley, "Extending Data-Driven Koopman Analysis to Actuated Systems," *IFAC-PapersOnLine*, vol. 49, no. 18, pp. 704–709, 2016.

[18] H. Arbabi and I. Mezic, "Ergodic Theory, Dynamic Mode Decomposition, and Computation of Spectral Properties of the Koopman Operator," *SIAM Journal on Applied Dynamical Systems*, vol. 16, no. 4, pp. 2096–2126, Jan. 2017.

[19] S. Peitz and S. Klus, "Koopman operator-based model reduction for switched-system control of PDEs," *Automatica*, vol. 106, pp. 184–191, Aug. 2019.

[20] M. O. Williams, C. W. Rowley, and I. G. Kevrekidis, "A Kernel-Based Approach to Data-Driven Koopman Spectral Analysis," Jul. 2015.

[21] S. L. Brunton, B. W. Brunton, J. L. Proctor, E. Kaiser, and J. N. Kutz, "Chaos as an intermittently forced linear system," *Nature Communications*, vol. 8, no. 1, p. 19, Dec. 2017.

[22] S. Van Huffel, "Iterative algorithms for computing the singular subspace of a matrix associated with its smallest singular values," *Linear Algebra and its Applications*, vol. 154-156, pp. 675–709, Aug. 1991.

[23] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, Apr. 2016.

[24] S. L. Brunton, B. W. Brunton, J. L. Proctor, and J. N. Kutz, "Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control," *PLOS ONE*, vol. 11, no. 2, H. A. Kestler, Ed., e0150171, Feb. 2016.

[25] H. Arbabi, M. Korda, and I. Mezic, "A Data-Driven Koopman Model Predictive Control Framework for Nonlinear Partial Differential Equations," in *2018 IEEE Conference on Decision and Control (CDC)*, Miami Beach, FL: IEEE, Dec. 2018, pp. 6409–6414.

[26] M. Hankaniemi and T. Suntio, "Dynamical Modeling and Control of Current-Output Converters," *International Journal on Energy Conversion (IRECON)*, vol. 7, no. 5, p. 197, Sep. 2019.

[27] Ashish Agarwal and P. et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[28] A. Paszke and S. e. a. Gross, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," Dec. 2019.

[29] P. Virtanen, R. Gommers, and Oliphant, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[30] I. Polat, *Harold: A control systems package for python3 systems*, Software available from tensorflow.org, 2020. [Online]. Available: https://pypi.org/project/harold/.

[31] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[32] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *Journal of Control and Decision*, vol. 5, no. 1, pp. 42–60, 2018.

[33] J. Tu, C. Rowley, D. Luchtenburg, S. Brunton, and J. Kutz, "On dynamic mode decomposition: Theory and applications," *Journal of Computational Dynamics*, vol. 1, Nov. 2013.

[34] B. R. NOACK, K. AFANASIEV, M. MORZY□SKI, G. TADMOR, and F. THIELE, "A hierarchy of low-dimensional models for the transient and post-transient cylinder wake," *Journal of Fluid Mechanics*, vol. 497, pp. 335–363, Dec. 2003.

[35] S. Klus, F. Nüske, S. Peitz, J.-H. Niemann, C. Clementi, and C. Schütte, "Data-driven approximation of the koopman generator: Model reduction, system identification, and control," *Physica D: Nonlinear Phenomena*, vol. 406, p. 132 416, May 2020.

[36] N. Takeishi, Y. Kawahara, and T. Yairi, "Learning koopman invariant subspaces for dynamic mode decomposition," *CoRR*, vol. abs/1710.04340, 2017.

[37] E. Kaiser, J. Kutz, and S. Brunton, "Data-driven discovery of koopman eigenfunctions for control," Jul. 2017.

[38] L. P. A.C. Hindmarsh, "Lsoda: Ordinary differential equation solver for stiff or non-stiff system," *Organisation for Economic Co-Operation and Development, Nuclear Energy Agency*, 2005.

[39] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds., Morgan-Kaufmann, 1992, pp. 950–957.

[40] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, Dec. 2014.

[41] I. Loshchilov and F. Hutter, *Sgdr: Stochastic gradient descent with warm restarts*, 2016.

[42] R. Ge, S. M. Kakade, R. Kidambi, and P. Netrapalli, *The step decay schedule: A near optimal, geometrically decaying learning rate procedure for least squares*, 2019.

[43] C. Wu, Y. Zhao, and Z. Wang, "The median absolute deviations and their applications," *Communications in Statistics - Simulation and Computation*, vol. 31, pp. 425–442, 2002.

[44] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *ICINCO*, 2004.

[45] H. Ayala, R. Sampaio, D. M. Muñoz, C. Llanos, L. Coelho, and R. Jacobi, "Nonlinear model predictive control hardware implementation with custom-precision floating point operations," in *2016 24th Mediterranean Conference on Control and Automation (MED)*, 2016, pp. 135–140.

[46] Y. Cho and L. Saul, "Kernel methods for deep learning.," Jan. 2009, pp. 342–350.

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," Jun. 2016, pp. 770–778.